MASTER IN COMPUTER SCIENCE PROFESSIONAL FOCUS IN SOFTWARE ENGINEERING

Automated mission-specific configuration for a multi-drone emergency response system

Van Speybroeck, Maxime

Award date:
2020

Awarding institution:
University of Namur

UNIVERSITE DE NAMUR

Faculté d'informatique
Année académique 2019–2020

Automated mission-specific configuration for a multi-drone emergency response system

Van Speybroeck Maxime

Maître de stage : Cleland-Huang Jane

Promoteur Heymans Patrick (Signature pour approbation du dépôt - REE art. 40)

Mémoire présenté en vue de l'obtention du grade de

Master en Sciences Informatiques.

Acknowledgements

I would like to thank each person who has been involved in realizing this thesis.

First of all, I want to thank Jane Cleland-Huang, professor at the department of computer and science engineering in Notre Dame, who accepted me in her research team and supervised all of my work there. I have learned many things from Jane and became a better student in Notre Dame.

The research team has been part of all the work I have been doing in Notre Dame, I want therefore to thank Nafee Md. Al Islam, Ankit Agrawal and Eric Tsai with whom I have been working every day toward my goal.

I also would like to thank my referent teacher in University of Namur, Mister Patrick Heymans for all the advices I received from him during my thesis writing.

I also want to thank every single person I met in Notre Dame that welcomed me for my first experience abroad.

Finally, I want to thank my family for its support during these 6 months of hard work.

Table of contents

PART I. Introduction

These days, Unmanned Aerial Vehicles (UAVs) or Drones are increasingly used for everyday tasks. They dispose of both hardware and software capabilities that allow them to be used in various application domains. Current examples of using drones include flying drones for hobbies, video streaming for sport events, carrying supplies and deliver them to a specific spot, human tracking many others.

Emergency responders are called every day for time-critical tasks and sometimes in hard to reach spots. UAVs are already used by emergency responders to support them in various tasks of their work. However, drone uses are typically limited to a single drone managed remotely by a human. University of Notre Dame is currently working on a project that aims to increase the use of drones in emergency missions. They envision using a cohort of drones acting semi-autonomously in order to support emergency responders. This however addresses a lot of safety concerns as well as many traditional software challenges.

This thesis has been written based on a four months internship at Notre Dame. During this time, the author of this thesis has worked on two main approaches to quickly eliciting and modeling requirements as well as configuring drones to achieve a particular mission. The first approach aims to configure a drone-based response system using verbal descriptions of missions while the second one comes with a configurator tool for a user to select an existing mission type or create a custom mission.

Work done in Notre Dame includes requirements engineering, programming, visualization tool use as well as academic research.

PART II. Background

Chapter 1. The DroneResponse Project

Every day, emergency responders are facing sensitive and complex situations in which human lives may be in danger. Firefighters are called for various tasks from rescuing someone drowning in a lake to securing an accident location or fighting a structural fire. Their response time is critical and therefore planning and acting efficiently is a must for them. Today's evolving technologies enable small unmanned aerial vehicles (named UAV's or drones) to be deployed alongside with emergency responders to support them in these time-critical tasks. However, their use is typically limited to a single UAV controlled manually by a human. Our goal is to create a system that manages not only one drone but a cohort of drones acting semi-autonomously as members of the rescuing team. This would free emergency responders of time-consuming tasks and would let them focus on high-level mission goals while drones would contribute toward the success of the missions executing lower-level tasks. Drone mobility is extremely useful to gather information about the environment in which emergency responders operate. Since emergency responders are often put at risk, this will help them achieve missions more efficiently but also more safely. UAV's and emergency responders might then be working collaboratively in order to improve the quality of response for missions. Indeed, information gathered by the drones will allow emergency responders to act in better conditions, knowing a lot more about the working environment, thus making the right decisions at the right time, picking the right equipment in advance and so much more.

Drones can be envisioned as supporting emergency responders for missions of different types. For example, they might be helpful to fully cover an assigned area of a lake where a victim is missing in order to locate it. They would provide visual information and accelerate the search. They may also be used to quickly create a 3D heat map of a building on fire, generate thermal imagery of the rooms and look for people trapped inside the building. Medical deliveries like transporting a defibrillator in the mountains where a person is stuck is also a potential use of the system. Other researches are already investigating the use of drones in many scenarios. One of them is using UAVs to search for victims and perform basic measurements on them (heart and respiratory rate) through a contactless method (Arias, et al., 2019).

Another one is using UAVs as a survey tool for topographic mapping and measurement in the coastal zone (Turner, et al., 2016). A researcher also use drones to deliver supplies to hard-to-reach locations like a defibrillator delivery to treat people suffering from a cardiac arrest (Fleck, 2016).

To enable all of these mission cases, there is a need for new conceptual models. Drones need way more autonomy than current practices in order to achieve such tasks. The level of autonomy of each drone should be adapted, for example in a search-and-rescue mission, if the drone may have found a victim, it might ask for the victim's confirmation while in some cases a drone with greater autonomy might take the decision itself without human intervention. Different challenges have to be addressed like runtime-adaptation, image recognition, goal modeling, situation awareness but also safety concerns.

The computer science and engineering department of University of Notre Dame[1], Indiana has developed an Open-Source project named *Dronology*[2]. It is a long-term project that has for goal to address several challenges in safety-critical software systems. *Dronology* provides a research incubator based upon a platform for managing and coordinating the flights of multiple physical / simulated UAVs (Cleland-Huang, et al., 2018). Leveraging *Dronology*, a research team of the computer science and engineering department of University of Notre Dame has launched the *DroneResponse* project. The purpose of this project is to develop a system for deploying several semi-autonomous UAV's alongside human responders during emergency missions. The system is built in order to address the challenges mentioned in the previous paragraph. It is designed in close collaboration with Notre Dame fire department to better understand emergency responders' needs, the application domain and to address situational awareness (Agrawal, et al., 2020). The author of this thesis has been involved in a 4 months collaboration with the research team on the *DroneResponse* project.

[1] https://www.nd.edu/
[2] https://dronology.info/

Because of the potential high number of mission types that we want to support, the main goal of this thesis is to come with a way to configure a mission efficiently prior to launch. This includes defining the mission and configure drones accordingly. Figure 1 illustrates a simple view of *DroneResponse* architecture. Yellow components will be configured according to the mission. Drones are imbued with a certain level of autonomy, various hardware and software capabilities as well as mission goals that has to be achieved. They communicate with the ground control station to exchange information about their environment or other UAVs. However, this thesis aims to configure the mission and drones at a pretty high-level. We therefore will not discuss how the system will then handle the configuration neither run-time adaptation but focus on **launch-time configurations**.

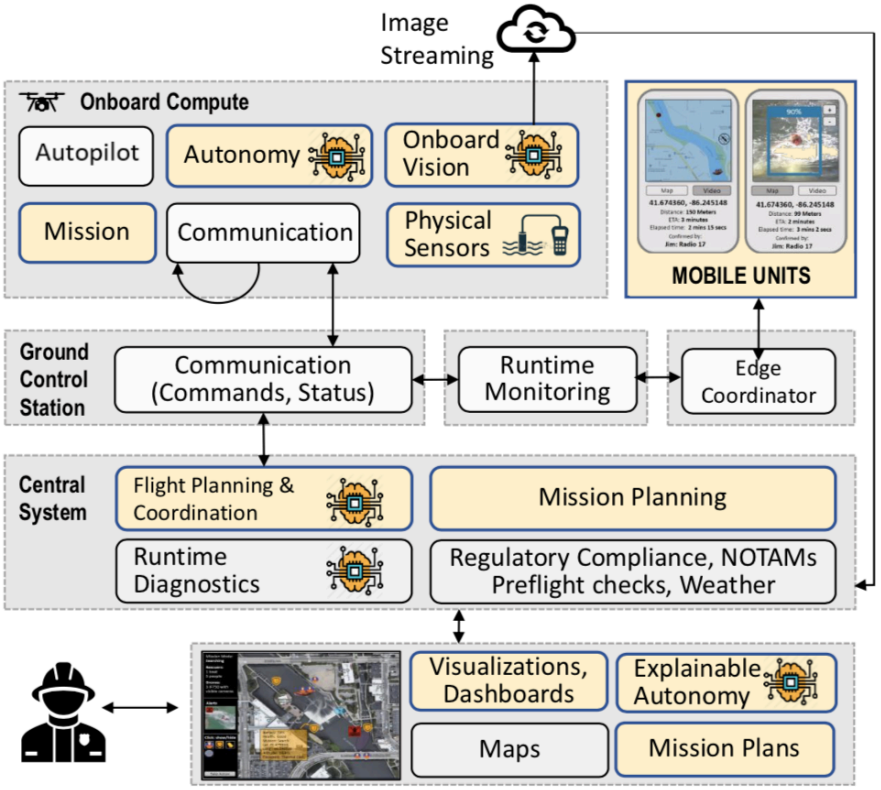


Figure 1: DroneResponse high-level architecture

We need to provide a way to configure acceptable missions (some tasks of a mission might be exclusive). In order to address this, we need a way to represent the different missions as a data structure including all the features of *DroneResponse*. The next chapter discuss Software Product Lines which is the way we build our system for our two approaches.

Chapter 2. Software Product Lines

Software engineering requires proper architectures and development processes in order to achieve the desired goals. We aim to build our *DroneResponse* as a product line which needs proper introduction.

In the past, in software engineering as well as is manufactures, companies were building products on clients' demand. However, as time passed, the number of clients to fulfill continuously increased. Facing this high demand, companies were spending too much time and money on building one product per customer. To address this problem, car manufacturers came with a way of producing goods in large quantities, called mass production. This way of producing allowed companies to create the same product several times easily but products could not be as diverse as before. At this point, companies needed a new way of building products that would combine the two previously described ideas which are building in large quantities and customize products according to the customers' wishes.

Companies started to use what is called common platforms, which means planning in advance which parts will be common to several products (Pohl, et al., 2005). A platform here can be seen as all the technological means used to build a product. The goal was to have as much of these technological capabilities in common to every product as possible. The artefacts built should be reusable, technologies and development processes be the same. Therefore, this allowed companies to reuse a common base of technology while still fulfilling individual customers' wishes without spending too much money on building products. Based on this we can now define the notion of **product line** which basically is *building products using the platform-based way while doing mass customization*. This way of building product also applies to software engineering which we call **software product lines**. Creating the platform is the most difficult part. Developers should focus first on commonalities then differences between the products. Flexibility is the key in software product lines, it is important in the development process to identify where products will differ so that artefacts can be built flexible enough. This kind of flexibility is named as variability in software engineering.

Basically, software product line engineering has many advantages for organizations that are building products with a common base for a lot of clients that still need some parts of the product customized. The first advantage is the reduction of development costs because of using reusable artefacts that provide less work building basics of each products. However, it requires an upfront investment for designing and building the artefacts the right way. Secondly, the overall quality of the products is improved through reusing components. The development cycles are shorter using software product lines. There are less important advantages like decrease of maintenance efforts, easier evolution and complexity management but the most important is a better quality at lower prices for the customers.

(Pohl, et al., 2005) describe a framework for software product line engineering. The framework is made of two main processes which are (1) domain engineering and (2) application engineering. Domain engineering refers to the process for creating commonalities and variabilities of the product line through producing the platform while application engineering relates to using the commonalities and variabilities defined in (1) to build applications of the product line. Figure 2 summarizes the two processes of the software product lines engineering framework according to Pohl.

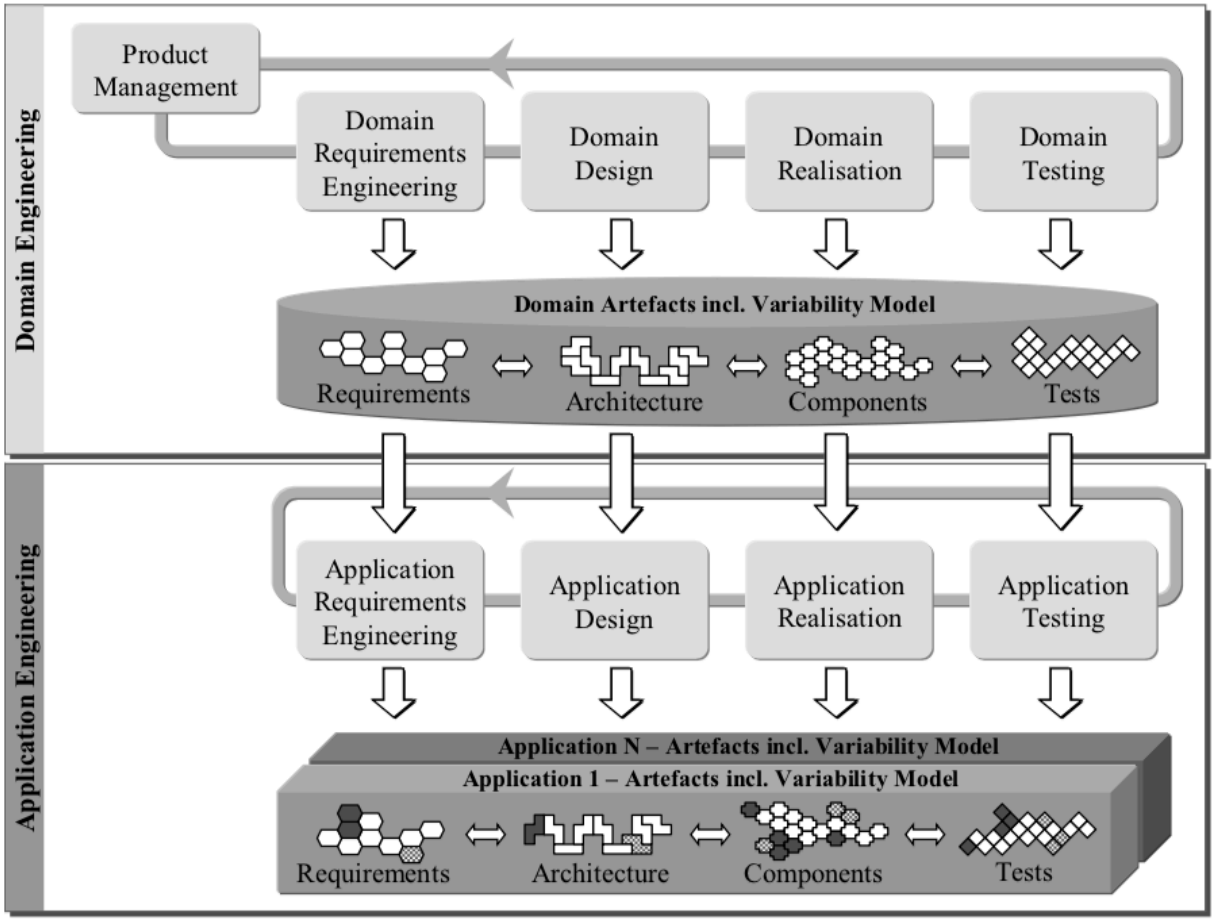


Figure 2: Software product lines engineering framework, extract from (Pohl, et al., 2005).

When creating a software product line, defining variability is a key concept. In this case, defining means identifying and documenting variability. This is done in the domain engineering process described previously. This thesis comes with two approaches for quickly configuring our *DroneReponse* system to achieve a desired mission. These two approaches have different requirement engineering processes to build *DroneResponse* as a product line. Therefore, we need a way to document variability in the requirement processes.

In this thesis, we use use-case diagrams and features models to describe variability. Features models have been introduced by (Kang, et al., 1990). A features model is a model that represent all the features of a system. Instantiating a product of the features model is done by combining mandatory, optional and alternative features while respecting constraints existing between the features. These diagrams are used to model software product lines and products one can create. This is therefore a good way to manage variability of software product lines through inclusion or exclusion of features.

Use-case diagrams written in this thesis have been written based on (Cockburn, 2000) and (Jacobson, 2004) work. The use cases in the two approaches are individual textual use cases representing mission scenarios. They contain a name, the actors part of the scenario, pre/post conditions as well as steps describing the scenario. Use cases from the first approach do not include exception cases. Use cases from the second approach tend to have common tasks shared across several scenarios, therefore, we also employ supporting use cases. These supporting use cases are used as references in the steps of the main use cases. This allows us to read use cases easier and keep consistency between common tasks of scenarios.

Use-case diagrams and features model are used in the two approaches to build the *DroneResponse* system as a product line which we tend configure in two different ways. The second approach also use activity diagrams in the requirement engineering process as well as to explicit missions to the user configuring it. Activity diagrams are useful to describe dynamic aspect of the system (Baresi, 2009). It represents the different activities happening in a scenario as well as the condition to move from one activity to another. The diagrams used are simple UML activity diagram.

Chapter 3. Related work

This chapter discuss the related work for our two approaches. Both approaches we introduce in the next chapters have for main goal to quickly configure our drone emergency system the right way for an emergency mission. However, the work put in place to achieve this starts in the way we build the system itself and so in the requirements engineering part of it. We therefore come with two different approaches to engineer the requirements and build our system as a product line.

The first approach aims to configure the feature model of the product line based on verbal descriptions. The mission descriptions we use are use-case diagrams giving details about the flow of events happening in a mission that is to happen. We are therefore discussing the ability to link the requirements of a system to some other models in order to manage and configure the system in the best way possible. A lot of work has been done in this domain which is transforming textual use cases into various form of models. (Guttierez, et al., 2008) and (Yue, et al., 2010) have been proposing a way to improve the visualization of textual requirements with approaches to transform use cases into activity diagrams. (Pudlitz, et al., 2019) shows how complicated it is to compare states of a system simulation to the requirements in order to verify that the system is compliant. Mapping the system states in natural language to the simulation states requires a deep understanding of the semantics of these states. They therefore propose a way to extract the different states of a system from its requirements specifications.

All of these works show great potential but our first approach is not related to the same concepts. Indeed, they come with ways to derive requirements into all kind of models (Activity diagrams, State machines and so on) but our first approach aims to map automatically the requirements described as use-cases steps to an **existing** feature model related to the system using a **classifier**. We therefore don't need techniques to understand the semantics behind the requirements of the system in order to build these other models.
(Bragança, et al., 2007) 's work might have been of great interest for this work. They have been proposing an approach that attempts to map use case diagrams to feature models in software product lines.

However, their method maps complete use case diagrams to a set of feature while our approach aims to map the different steps of each use-case diagrams to a set of features and not the entire use case itself. Moreover, their work is done by working on the models (use cases and features models) and adapting them while we use a classifier to do the mapping.

The second approach has a much larger scope both in term of requirement engineering and concepts used in the configuration part. Different topics are discussed along this approach including:

a) Use cases and variability
b) Feature model mapping
c) Product derivation and configuration
d) Multi-agent task specification

The last one relates the use of multi-agent robots which linked to the underlying mechanisms of *DroneResponse* and therefore not in the scope of the thesis author's work. This part will therefore focus on the related work on the three other ones.

The first topic discussed in the second approach (a) relates to use cases and their variability. (Pohl, et al., 2005)'s work, discussed in the background part, comes with a way of describing variability into their requirements-driven approach. (Bühne, et al., 2006) came with a scenario-based approach to describe variability. However, they used use-case diagrams in their requirement engineering process while we use textual use cases to describe our mission scenarios. We then use them to create appropriate models and merge them into a software product line. We could have used (Bertolino, et al., 2006)'s way of specifying use cases. This work presents an extension to the standard use case notation allowing to specify product line use cases. However, this did not match our approach of avoiding to think the system as a product line too early in the requirement engineering process.

Features model mapping (b) has been quite investigated by several authors. Just like in the first approach, we can cite (Bragança, et al., 2007) that proposed an automated mapping from use-case diagrams to features model. (Favaro, et al., 1998) integrated feature modeling in a Reuse-Driven Software Engineering approach which is a use-case driven reuse process. They provide a way for creating a feature model to support domain engineering.

These concepts are common to our approach but we provide a configurator tool for deriving products from our product line.

Finally, related work for (c) is also different than the approach we used for creating a configurator tool in order to configure our *DroneResponse* system built as a product line. (Briand, et al., 2016) created a tool that guide the user through a process for configuring products from product line models. However, it is only built to work with variability modeling for artefacts that are common to product lines like use case diagrams. This approach has great potential but the authors use an opposite process to our approach. Indeed, while we start from use cases and build intermediary models to end up with a product line, they aim to generate product specific use cases based on the product created with their tool. Our tool, in contrast, generate mission-specific configuration and activity diagram related to the mission. A few other researches have been investigating product derivation in software product lines but our approach provides a configurator tool based on specific use cases scenario built with stakeholders in order for them to configure the associated product line.

PART III. Contributions

This chapter explains the two approaches we came up with to quickly elicit requirement and configure our *DroneResponse* system for a specific mission. The two approaches differ both in the requirement engineering process and the way we attempt to configure drones and the system to achieve a particular mission. We ended up writing one research paper for each approach. While the paper related to the first approach is to be submitted, the second one has been published and accepted for the Software Product Line Conference 2020. The author of this thesis has been working mostly on the configurator tool of the second approach, as well as the entire process of the first approach. Therefore, the requirement engineering process described in the second approach is based on the paper written by the research team (Cleland-Huang, et al., 2020).

Chapter 4. Verbal mission statement approach

In this approach, we model our system as a product line composed of both hardware and software features. The goal is then to configure drones prior to launch in order to perform the right actions corresponding to the mission and to equip them in advance with appropriate hardware materials. Indeed, some mission types require specific physical hardware that not all drones have on board. For example, environmental sampling missions would probably need some sampling sensors on the drones performing the mission. Since the main goal of this paper is to quickly elicit requirements for a mission, this approach has for objective to configure the product line based on verbal statements of the mission described by an emergency responder.

The following sections detail the specific requirement engineering process used in this approach. We collect a set of 20 use-cases in which we tag every step. The tagged use-cases are then used to train a classifier. Each of the tags used in the process are then matched to features of the product line. The goal here is, for a previously unknown use-case, that the classifier automatically tags all of its steps in order to select the corresponding features on the product line feature model.

4.1. Use-cases elicitation and tagging process

The information on which the 20 use-cases are built come from publicly available scenarios, conversations with lead inventors of a UAV medical delivery start-up company, brainstorming sessions with the South-Bend fire department and external resources. The use-cases are summarized in Figure 3. The complete set of use cases can be found in the Appendices part.

ID	Use Case Name	Knowledge Source
UC1	River Search-and-Rescue	Emergency responders
UC2	Defibrillator Delivery	DeLive spinoff CTO
UC3	Traffic Accident Monitor.	Emergency responders
UC4	Chemical spill	Online resources
UC5	Avalanche Rescue	Online resources
UC6	Suspect tracking	Online resources
UC7	Burning building	Emergency responders
UC8	Water Sampling	Environmental Scientists
UC9	Air Quality Monitoring	Online resources
UC10	School shooting	Team brainstorming
UC11	Radiation Detection	Emergency responders
UC12	Man overboard	Discussions with Navy
UC13	Crowd control	Team brainstorming
UC14	Underwater Ice rescue	Online resources
UC15	Flood support	Online resources
UC16	Map areas after earthquake	Online resources
UC17	Rip current rescue	Team brainstorming
UC18	Lost kayaker	Team brainstorming
UC19	Volcanic eruption	Online resources
UC20	Storm utility inspection	PrecisionHawk.com

Figure 3: Table of the twenty use cases built in the requirement engineering process

Each use-case includes a name, list of actors, pre-conditions and a set of sequential steps describing the main scenario of the mission.

Four different use-cases are tagged using an inductive approach. One tag is assigned to each step of each use-case. Because of this “one step one tag” approach, some steps describing more than one actions have to be split in several steps. This process leads us to 50 candidate tags. The research team discusses these tags altogether in order to refine names or the level of abstraction of each tag. The main difficulty for this particular process is to achieve the same level of abstraction for each tag. Indeed, some steps are very specific while other ones were pretty high-level and common to a lot of use-cases. Once we agree on this set of tags, we use them to tag the remaining use-cases. We notice that 6 steps of these remaining use-cases don’t match any of the agreed tags. Therefore, three additional tags are added to the set of tags.

Figure 4 shows the final set of tags used for the 20 use-cases.

The "tag" column shows the name of the tag while the "Cnt" column shows how many time a tag appeared in the 20 use-cases steps. An example of a tagged use-case for creating and updating a heat map of an on-fire building can be found in the appendices part of this document (see Appendice 1).

Tag	Cnt	Tag	Cnt
Search And Survey	23	Return To Landing Spot	22
Fly to location	19	Take Off	19
Routes Planning	18	D2H Event Notification	16
Human Physical Activity	16	Scene Reconstruction	16
Image Streaming	15	Scene Annotation	12
Target Detection	11	H2D Communication	10
Delivery Capabilities	9	Vision Capabilities	9
Target Tracking	8	Drones Transported to Site	8
Remote Mission Launched	8	Environment Sampling	7
Mission Configuration	7	Camera Activation	6
Emergency Call	6	GPS Coordinates specified	6
Equipment added	6	Geolocation	5
Flight Prohibition	4	Drones Activated	3
Displaying Information	3	Target Found	3
Drone Monitoring	3	Collision Avoidance	2

Figure 4: Final set of tags used in the tagging process and number of time each tag appeared.

4.2. Using a classifier to label use-cases steps automatically

We use python library *SKLearn*[3] to train a classifier. Three different algorithms are used during the process:

- Random Forest (RF)
- Stochastic Gradient Descent (SGD)
- Logistic-regression (LR)

We feed the classifier with 19 tagged use-cases and test it on the last one then repeat the process so that every use-case is used as test case.

[3] https://scikit-learn.org/stable/

Tag Name	#	RF	SGD	LR
Search and survey	23	0.35	0.26	0.35
Return to landing spot	22	0.77	0.77	0.82
Fly to location	19	0.68	0.47	0.47
Take off	19	0.84	0.79	0.79
Routes planning	18	0.61	0.50	0.50
Human physical activity	16	0.63	0.56	0.50
D2H Event notification	16	0.63	0.69	0.63
Scene reconstruction	16	0.38	0.38	0.44
Image streaming	15	0.87	0.80	0.87
Scene annotation	12	0.50	0.58	0.58
Target detection	11	0.27	0.73	0.55
H2D communication	10	0.60	0.70	0.70
Delivery capabilities	9	0.33	0.33	0.44
Vision capabilities	9	0.44	0.67	0.67
Target tracking	8	0.25	0.25	0.25
Remote mission Launched	8	0.75	1.00	1.00
Drones transported to site	8	0.50	0.75	0.38
Mission configuration	7	0.29	0.43	0.29
Environment sampling	7	0.14	0.29	0.14
Emergency call	6	0.50	0.67	0.83
Equipment added	6	0.00	0.33	0.50
Camera activation	6	1.00	0.83	1.00
GPS coordinates specified	6	0.00	0.50	0.33
Geolocation	5	0.20	0.20	0.20
Flight prohibition	4	0.50	0.50	0.50
Displaying information	3	0.00	0.33	0.33
Drone monitoring	3	0.00	0.67	0.67
Drones activated	3	0.00	0.00	0.00
Target found	3	0.00	0.00	0.00
Collision avoidance	2	0.00	0.00	0.00
Weighted average		**0.51**	**0.55**	**0.55**

Figure 5: Accuracy by tag for each of the three algorithms

Figure 5 presents the accuracy obtained by tag with the three algorithms. SGD and LR algorithms have a better total accuracy than RF. Accuracy seems to increase when the frequency of a tag goes up but the most frequent tag which is "*Search And Survey"*, appearing 23 times has a low accuracy (0.35) therefore breaking the rule. This probably comes from the level of abstraction of the tag. Indeed, "*Search And Survey"* applies to a lot of different search functionalities. Discussing again the set of tag and adding lower-level tags might help the results getting better.

We then calculate the harmonic mean of recall and precision (also called F1 Score in statistics) for classifying each step of all use-cases with the three different algorithms. Again, SGD and LR outperformed the Random Forest algorithm. We thus decide to go on with the Stochastic Gradient Descent algorithm for the experimentations since it has the best results with Logistic-regression but it got better results on previously unseen data. Figure 6 presents the F1 Score achieved by the classifier on our twenty use-cases using SGD algorithm. The last two points on the right are results for the experimentation we discuss in section 4.4 of this document.

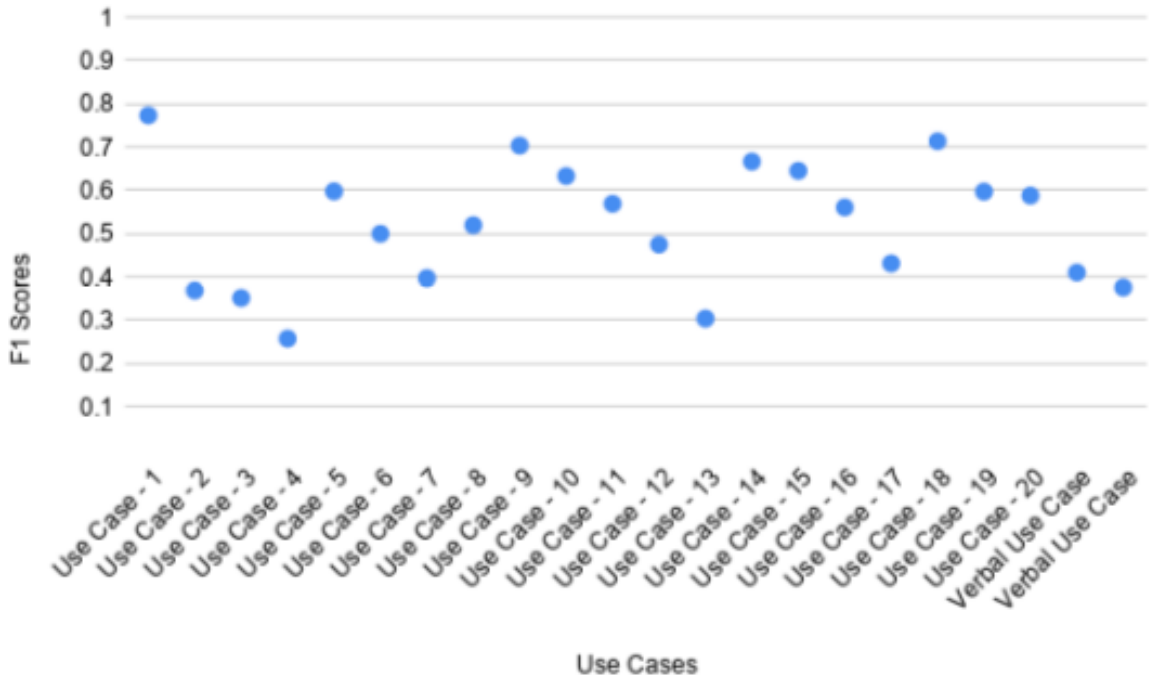


Figure 6: F1 score of the classifier for tagging 20 use cases and 2 test cases using SGD algorithm.

4.3. Feature model creation and tags mapping

This approach is based on existing and currently in development models and systems. *DroneResponse* product line includes a feature model, mission-related goal-model, a product line architecture supporting all valid configurations and so on. However, since the main goal of this document is focusing on quickly eliciting requirements to specify a mission and identify the corresponding features of the product line, we focus on the features part of the system.

Based on the set of tags from the previous steps, the search team creates a high-level feature model supporting all 20 use-case scenarios. We then verify that the concept transported by each tag can be found in one or more features of the feature model. In some cases, several tags are referring to the same feature while in other cases, a single tag refers to several features.

The model configured for the burning building heatmap use-case is depicted in Figure 7. Gray boxes represent mandatory features, also called commonalities to all products. White boxes represent optional features that might be selected or not for a specific product. Note that orange boxes on this graph are also optional features but that have been activated to fit this particular mission.

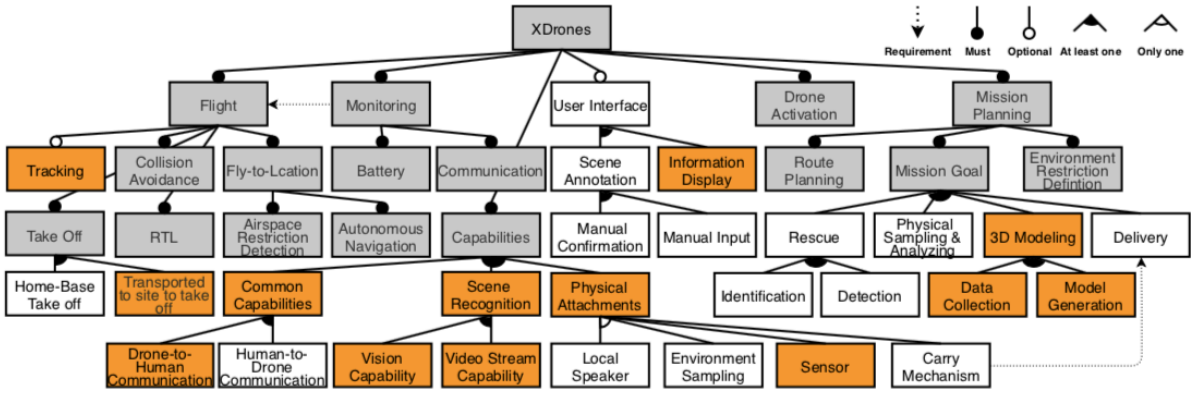


Figure 7: Features model built in the requirement engineering process and configured for a burning building heat map scenario.

Since a product line is composed of shared artefacts as well as artefacts specific to concrete products, these are directly connected to the feature model of the product line which represents the different mandatory and optional feature as well as constraints. It is important to understand that this document aims to select adequate features for a mission and configure drones prior to launch. Indeed, drones will probably have to adapt their behavior during the mission. This is why, our *DroneResponse* system is built a way that each drone will be able to understand changes in the current environment and therefore modify its behavior to fit the new mission state. This requires proper run-time adaptation models in the *DroneResponse* architecture but our work is not affected by all of these mechanisms since we focus on initial pre-launch configurations of the drones.

Using the generic features model just built, we manually create a mapping from every tag to its associates features. As an example, we map the tag "*Route Planning"* to the feature "*Route Planning"* and the tag "*Image Streaming"* to "*Video Stream Capability"* feature. This will allow us, from a new tagged use-case, to automatically select the right features on the feature model.

4.4. Approach evaluation

We evaluate our approach for (1) appropriately selecting the right features to configure drones and (2) quickly eliciting requirements for an emergency mission. The first one is evaluated by demonstrating the mapping of an example use-case and the second one by assessing the ability of our classifier to tag steps from two verbal mission descriptions.

1. Figure 7 shows the configuration for the use-case scenario of creating and updating thermal map of a burning building. Grey features are common to all products and orange ones are the optional features that have been activated for that particular mission using the mapping from the tags of the use-case steps to the features.
2. This second study of this chapter which was quickly eliciting requirements for an emergency mission is done in collaboration with a researcher who had prior experience working with UAV's for emergency response missions. We show the researcher one of our use-cases which is "River-search and rescue" as a model. He is then asked to verbally describe two missions that are
    a. Border surveillance
    b. Searching hikers lost in the mountains

When an emergency call is received, the alpine rescue service is dispatched to find and retrieve the person that got lost in the mountain. [Human_physical_activity]

They carry a number of drones as part of their equipment in the backpack to the area where the person is suspected. [drones_transported_to_site]

Once the rescuers are in the general area where the person is reported the drones are unpacked and prepared for takeoff. [take_off]

Drones are equipped with cameras and need to have advanced object avoidance hardware, for example, Lidar, ultrasonic or cameras to detect and avoid obstacles such as trees, mountains etc. in hilly and wooded areas. [camera_activation] [equipment_added]

Depending on the severity of the situation critical medication or a first-aid box can also be loaded onto one of the drones.[equipment_added]

The target search area is selected by the rescuers and the drones take off and coordinate their search areas. [routes_planning][take_off]

The drones search the area looking for the missing person using image recognition and if available cellphone and/or GPS tracking. [search_and_survey][vision_capabilities]

When the person is detected or something is recognized by the drones on the images they are analyzing, the location and video stream is reported to the rescuers and they get notified on their mobile device. They can then confirm that the person was found by the drone found or reject the sighting as a false-positive. [target_found]

Figure 8: Verbal description of the "Seaching hikers lost in the mountains" scenario.

The researcher is constrained to less than three minutes for each description. The mission description provided for (b) is depicted at Figure 8.

To analyze the results, we separate the text verbally described into sentence-like chunks. The research team then discuss the chunks and manually tag each of them with one to three tags. Following this, we run the SGD classifier to automatically tag the chunks identified. If the classifier matches one of the tags from the research team, we count the prediction as correct. Green tags on Figure 8 shows correct prediction for that particular test case. The results can be found on the two points to the right of the graph showing the F1 scores of the classifier (see Figure 6). We see on the graph that the classifier achieved an F1 score of 0.36 and 0.40 for the two verbally described mission descriptions.

As a conclusion, we recognize some strengths in our first approach but improvements might get the results even better. First of all, the two verbal statements were grouping several ideas in one sentence while our 20 written use-cases INSERT 20 DS ANNEXES were split in a way that each sentence was related to one action of the mission. We might then improve our approach by using a classifier allowing several tags per sentence. We would also need way more use-case scenarios to train the classifier with way more data in order to improve the accuracy of the automatic tagging process. Finally, the definition of the set of tags should also be revised. Indeed, the level of abstraction of the tags has sometimes been an issue. Trying to get the same level for each of the tag is difficult since some steps of the scenarios are more specific than other one. Also, although results for activating the right features of the features model based on the mapping of a new use-case are encouraging, this is just a part of configuring a product. We therefore need more in order to configure drones and components of *DroneResponse* architecture. The second approach discussed in next sections links our requirements to concrete components, thus helping to configure the system properly.

Chapter 5. Configurator tool

Based on the lessons learned from the first approach and on previous experiences of the research team, we decided on moving to another approach to both mission configuration and underlying requirements engineering. In this approach, we use both use-case diagrams, features models but also activity diagrams to explicit the different steps of each mission. Our main goal in this approach is to build *DroneResponse* system as an easily configurable product line. However, this one comes with a stronger requirements engineering approach that will link the requirements for a mission to a configurator tool in order for the emergency responders to easily configure a mission that is to happen.

5.1. Requirements-driven approach to software product line creation

As introduced in Chapter 2, product lines are based on commonalities and variabilities, which are represented by mandatory, optional and alternative features. Creating a product of the product line means combining a set of these features. Since we are focused on the requirements elicitation part of the product line, the following text illustrates the way we created the system-wide feature model and the activity diagram for our product line. These two diagrams provide useful and concrete information to understand the different tasks of each possible mission scenarios as well as constraints between features.

As a reminder, we aim to create a configurator tool that will allow emergency responders to configure different types of missions. Two types of configurations should be configurable by responders:

- Known mission. This means that the scenario of the mission is common and already known. Only specific configuration points are left to configure. Configuration points are introduced in section 5.2. Examples of configuration points include defining a vision model for a mission, define a specific area for drones to search and so on.
- Creation and configuration of new mission. Emergency responders should be able to create new types of missions by associating different features of the product line and configure them to start the resulting mission.

The requirements engineering process of this approach is depicted at Figure 9. It starts by eliciting and modeling a family of related use cases. Based on these use cases, we construct individual feature models and activity diagrams for each of them and later merge them into a product line.

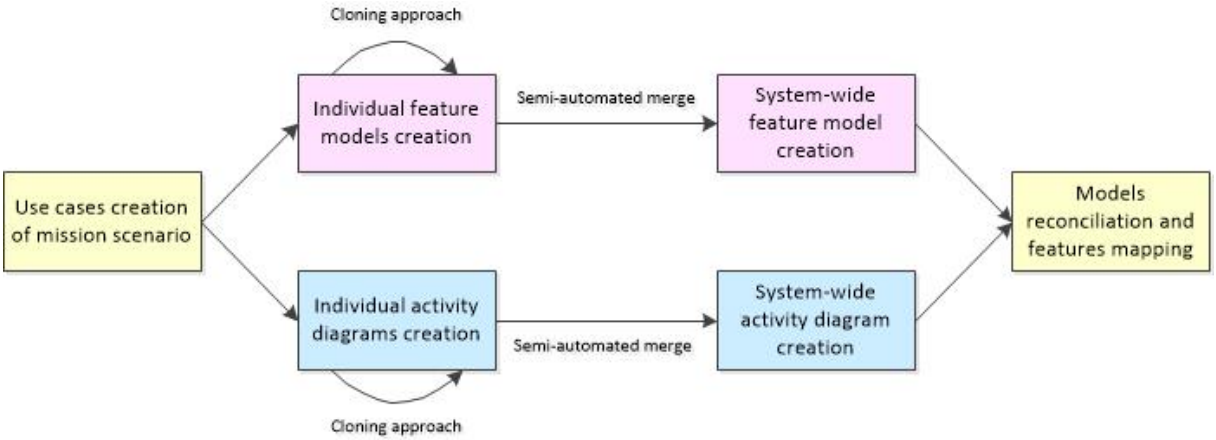


Figure 9: Requirements engineering process used to build system-wide feature model and activity diagram

During this approach, we use different sources of information to discover the most precise requirements of missions. First of all, we use the existing *Dronology* platform (Cleland-Huang, et al., 2018) which is the main building block of *DroneResponse* system. *Dronology* helped with existing architecture in which features and requirements from previous requirements engineering work are already part of. We also use literature (both academic research papers and grey literature) relative to "Drones" and "Emergency Response" to find existing descriptions of emergency responders using drones. We used this latter source not to find already existing use cases of emergency missions but to get various descriptions of several mission types and gather detailed steps about these mission scenarios. Sources are listed in Figure 10 The following subsections of this chapter detail our requirements modeling processes.

5.1.1. Use-case elicitation

We build 7 community-inspired use cases that will be used in the next steps for creating the product line. We aim to have some diversity in the use cases by defining different types of missions needing different types of software features, hardware capabilities, UAV autonomy levels and in which sequences of tasks will diverge. Each use case contains actors and stakeholders involved, pre/post conditions, the main success scenario describing tasks in a sequenced way as well as a set of exceptions. The use cases are built in an iterative way. First of all, we start with the "River search-and-rescue" scenario using our existing specifications (Agrawal, et al., 2020). Then, we clone this first use case and adapt it to the next mission scenario that is "Ice rescue". Adaptations include adding, deleting and modifying steps of the use case. We follow the same approach to complete all 7 use-cases, cloning each time the most similar use case to the next one that is to build.

At this point, we only focus on gathering requirements, common tasks and practices of individual emergency mission scenarios. This allows us not to think with a system view at this point of the process.

Figure 10 shows the 7 use cases built in this process as well as the sources and stakeholders used to create them. These 7 mission scenarios are the known missions discussed in section 5.1. which means that they are common and will be ready to pick in the configurator tool (although configuration points still might be configured).

Use-case ID	Use-case name	Sources	Stakeholders involved
UC1	River search & rescue	(Silvagni, et al., 2016)	South Bend Firefighters
UC2	Ice rescue	(Rios, 2019)	/
UC3	Defibrillator Delivery	(Fleck, 2016), (Mesar, et al., 2018)	DeLive, Cardiac Science
UC4	Traffic Accident	(Molino, et al., 2016), (Padua, et al., 2020)	South Bend Firefighters
UC5	Structural Fire	(Griffith, et al.)	South Bend Firefighters
UC6	Water Sampling	(Koparan, et al., 2018), (Lally, et al., 2019)	Environmental Scientists
UC7	Air Sampling	(Alvear, et al., 2015), (Chang, et al., 2016)	Environmental Scientists

Figure 10: 7 Use cases built representing typical missions for our DroneResponse system

Use Case: Ice Search and Rescue
ID: SPLC-11

Description
UAV(s) dispatched with a flotation device for ice rescue

Primary Actor
Drone Commander

Trigger
The Drone Commander activates the delivery.

Main Success Scenario
1. Emergency responders **plan_area_search** [SPLC-1001]
2. The DroneResponse commander issues a command to start the mission.
3. The UAV(s) **takeoff** [SPLC-1007]
4. The UAVs **perform_search** [SPLC-1002]
5. The UAV **requests_victim_confirmation** [SPLC-1005] from the human operator.
6. The UAV receives confirmation from the human operator that the victim sighting is valid.
7. DroneResponse automatically sends the GPS coordinates to the mobile_rescue system.
8. The UAV switches to **flotation_device_delivery** [SPLC-1006] mode.
9. Human responders reach the victim's location and execute a rescue.
10. The Drone Commander **ends_mission** [SPLC-1007].

Specific Exceptions
1. In step 3, one of the UAVs fails to take-off.
   1.1 If a replacement UAV is flight-ready, it is dispatched in place of the failed UAV.
   1.2 If no replacement is available DroneResponse re-executes **generate-search-plan** [SPLC-1009] for the available UAVs and previously defined search area.

General Exceptions
1. At any time, if communication is lost between the Ground Control Station and a UAV, DroneResponse executes the **Lost Drone-to-GCS Communication** (SPLC-2001) exception case.
2. At any time, a malfunction error is raised by a UAV in flight, DroneResponse executes the **Drone-in-flight Malfunction** (SPLC-2002) exception case.

Figure 11 UC2 use case created for an ice search-and-rescue scenario

Figure 11 shows UC2 use case which is an ice search-and-rescue scenario.

As described in the background section of this thesis (see Chapter 2), we used textual use cases to describe mission scenarios. We use supporting use cases to group common tasks while specific tasks are written in the main success scenario, therefore improving visibility for each use case. The remaining use cases can be found at [4].

[4] https://tinyurl.com/ybqq4ut2

5.1.2. Mission-specific feature models construction

For each mission scenario created in the previous subsection (7 use cases), we create individual feature models. Instead of creating a system-wide feature model that supports each of the mission scenarios, we use a bottom-up approach starting with individual feature models of each scenario. Features needed for each of the mission scenarios are identified manually then composed into a hierarchy of mandatory, optional and alternative features. We start with the “River search-and-rescue” scenario (use case 1). We then follow the same cloning approach as the one used with the use cases, for the next feature model to build, we clone the most similar one and adapt it to fit the mission scenario.

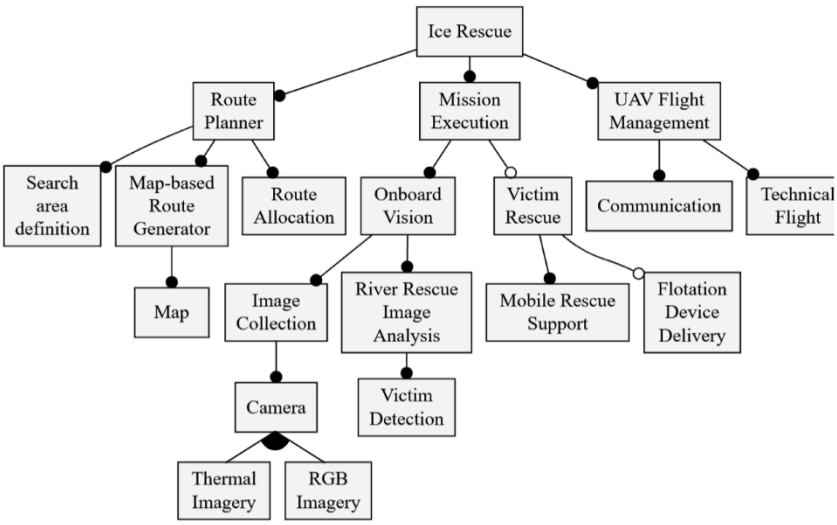


Figure 12 Individual feature model for the mission scenario "Ice rescue" (UC2)

Figure 12 shows the individual feature model created for the mission scenario “Ice rescue” (use case 2).

5.1.3. Semi-automated merge of individual feature models

The next process of the requirements modeling part is a semi-automatic merge of individual feature models into a product-line level feature model. We use an incremental technique to limit the complexity of this part of the work. Indeed, merging seven diverging feature models into one without encountering any constraints would not be possible. We thus start with the “River search-and-rescue” feature model as a baseline and use a simple automated name-matching algorithm to merge the next mission scenario feature model into the baseline. Because of the clone-driven approach used in the creation of both use-case diagrams and feature models, the name of the features matched in most of the merges. However, we have to inspect and refine the resulting model after each merge to ensure integrity of the model.

In a few cases, names did not tend to match because some words differed from other feature models.

The technique for selecting the next feature model to merge is opposite to the one used for use cases and individual feature models cloning. Here, we select the less similar feature model to merge into the main feature model in order to address major structural differences very early in the creation of the product line. The resulting product-line level features model is depicted at Figure 13. We color the features that belong to a specific mission type in order to improve visibility of the graph. Green features are used in environmental sampling scenarios (UC6, UC7), blue features in rescue scenarios (UC1, UC2), yellow features in delivery scenarios (UC3). The rest of the features (grey ones) are used in multiple scenarios.

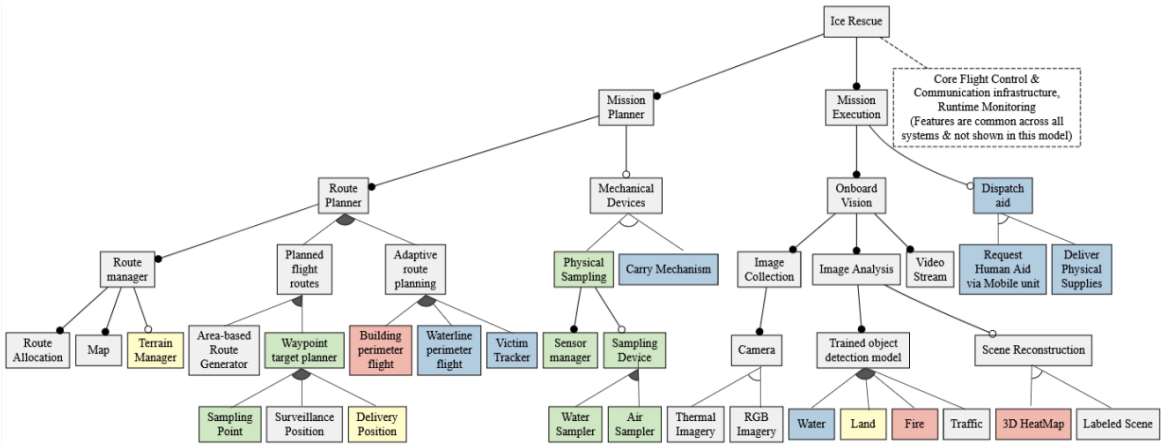


Figure 13 Merged feature model of the second approach.

5.1.4. Mission-specific activity diagrams construction

Each mission scenario is represented by a set of tasks, occurring in a specific order, sometimes according to conditions. Activity diagrams are used to document the flow of activities of a system, in order to understand the sequential part of it. Activity diagrams are pretty useful because they are high-level and therefore easy to understand by the stakeholders. We intend to use activity diagrams for two purposes that are (1) communicating the mission emerging from the configuration using the configurator tool discussed in the next chapter. This allows emergency responders to have a summary of the mission they pick or configure. We also intend to use activity diagrams for (2) visualize the state of the mission and display what tasks are performed by each drone during the mission. The second use is however not discussed further in this document but is future work for this project.

We constructed each mission-specific activity diagram manually using the same cloning approach as the feature model process (cloning the most similar one each time).

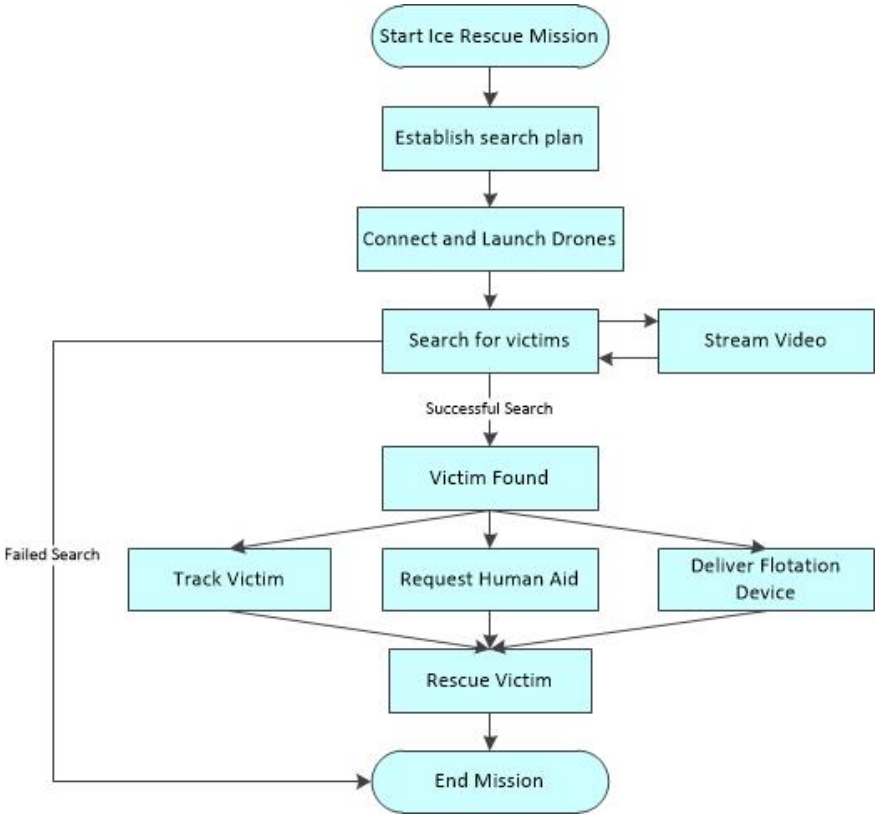


Figure 14 Individual activity diagram for an "Ice-rescue" scenario (UC2)

An example of the activity diagram created for the “Ice rescue” scenario (use case 2) is depicted at Figure 14. The difficulty of this part of the work is to find the right level of abstraction for the activity diagrams. Too high-level activity diagrams don’t show specific configuration points of the missions. Examples of these are defining an area versus precise coordinates for the drones to perform a search or which analyzer to use in a sample mission. Too low-level activity diagrams can get emergency responders lost into visualizing too many information about the mission.

5.1.5. *Semi-automated merge of activity diagrams*

The same approach as the semi-automated merge of feature models is used to merge activity diagrams (incremental approach using a simple name-matching approach with the less similar activity diagram available). Again, a few nodes for which the name-matching algorithm did not work were refined. The resulting model is shown at Figure 15.

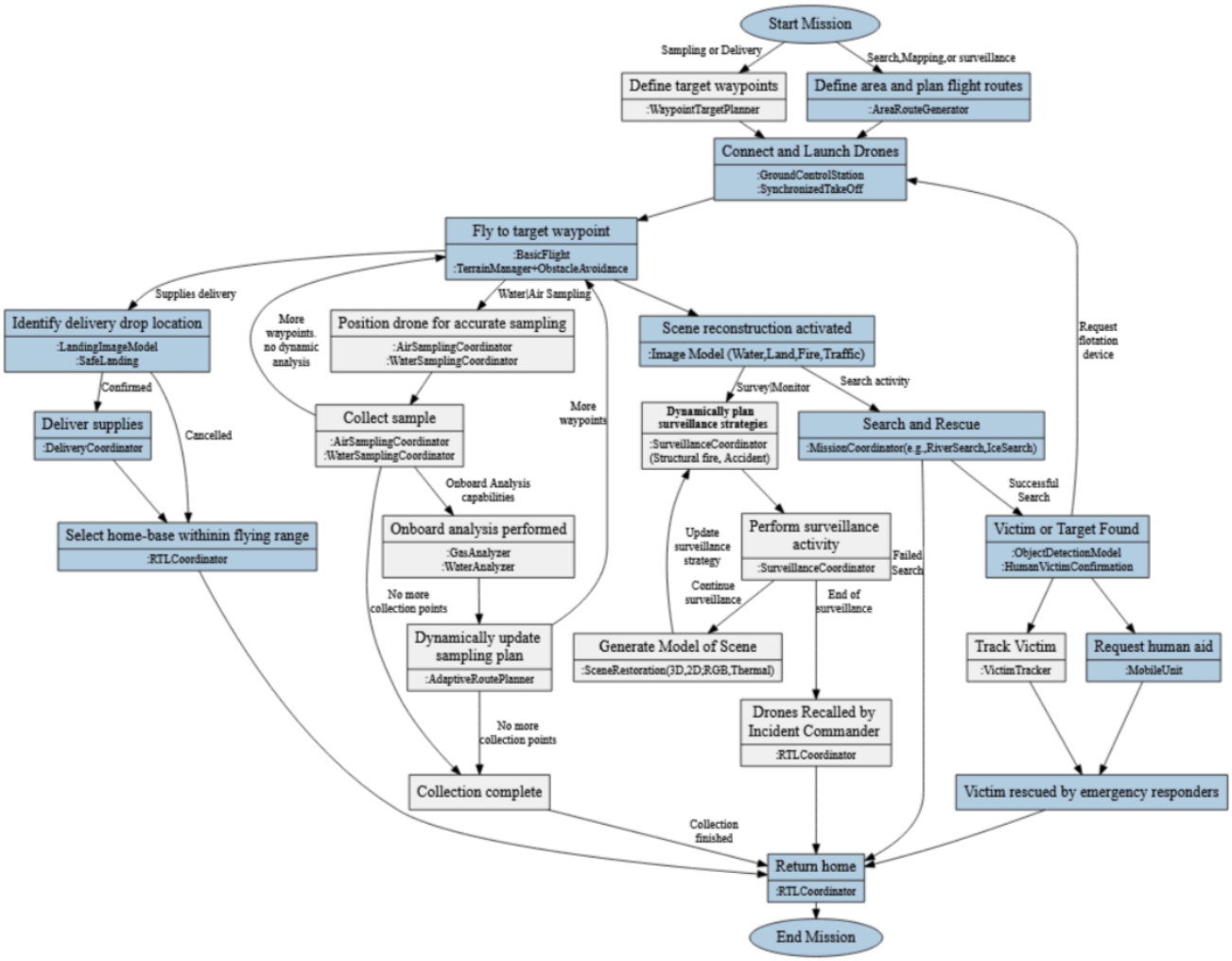


Figure 15 Merged activity diagram of the second approach

5.1.6. Reconciling models and mapping tasks to features

This last step of the process might be the most important one. As a matter of fact, in order to configure missions, we need to link all of these models to components of the *DroneReponse* architecture. The research team performs a manual mapping from each feature of the merged feature model to concrete components of the *DroneResponse* implementation. Some components already exist while some still are to be built. Then, the team performs a mapping from the merged activity diagram nodes to the features of the system-wide features model. Therefore, concrete component of the *DroneReponse* architecture can be mapped to nodes of the merged activity diagrams. Each node of Figure 15 shows the component it is related to in the lower part of the node with the notation ":ComponentName". This step is really crucial because it is the building block on which the configurator work lays. Indeed, the primary goal of the paper is to quickly and easily configure mission scenarios for emergency responders. Thus, the use of models like activity diagrams and feature models eases the configuration for the responders because they are really simple to understand and manipulate. However, we needed a strong relation from these models to concrete components in order to configure the system properly.

5.2. Configuration process

We engineered our *DroneResponse* system as a product line to get the configuration process as efficient as possible for the emergency teams that will use it. Indeed, emergency responders act most of the time under pressure in time-critical scenarios. Therefore, they need to be able to configure missions pretty fast, without too much effort and avoiding to check whether mistakes were made or changing the configuration of the missions. While the first approach aimed to configure drones using verbal statements to describe a mission that is to happen, this one, thanks to the various models we built in the requirement process, aims to provide a useful configurator tool. It is using merged activity diagram and features model that we present our *DroneResponse* configurator tool. We first of all detail the configuration process that emergency responders will go through then discuss the concrete implementation of the tool, discussing technological choices and the code. Finally, we realize a user study using the tool in order to evaluate our approach and draw conclusions.

Our configuration process consists of four steps that follow each other. At the end of these steps, the configuration tool outputs a mission specification in JSON format and generates an activity diagram that represents the mission configured. This allows responders to have a summary of the configuration and send the mission specification to the back-end service to execute the mission.

i. First of all, the user is asked to either select an existing mission or to configure a new mission type. This is what we named as “known mission” versus “new mission” in section 2.1. Existing missions are common missions that emergency responders will have access to without needing further configurations. However, although the mission tasks and sequences are already defined, it is still possible to configure low-level components that we call configuration points. These are discussed in the following paragraph. For now, the existing missions are the ones listed in section 5.3.3.
ii. Second, according to the previous choice, the user either (1) visualizes an activity diagram representing the mission if they picked an existing mission or (2) is asked a series of questions if they chose to create a new mission. In this latter case, the question answers are linked to requirements and the corresponding activity diagram of the new mission is gradually built after each user answer. For each new mission scenario, there is a maximum of 5 questions asked to the user. These questions are shown in the implementation part of this chapter, section 5.3.1. The user ends up in both scenarios with a complete activity diagram detailing the flow of the mission.
iii. The third step of the configuration process is the configuration of specific component of *DroneResponse* system (see section 5.3.3). We decide on decorating activity diagram graph nodes that require configuration with icons. Each node of the built graph that is linked to a concrete component in *DroneResponse* architecture has an icon to the right of the node. The user can click on an icon to open a modal window in order to configure that particular component. All configuration points have default values hence users can directly rush to the next step if needed. Examples and details about configuration points are detailed in the next subsections of this paper.

iv. Finally, some runtime configurations are proposed to the user. Some are asked to the users while others are configured in the back, independently of the user. An example of runtime configuration including the user is defining the autonomy level of drones for that particular mission. Hence, according to the autonomy level, drones with higher autonomy level might decide themselves on tracking a potential victim while drones with lower autonomy level would ask for human confirmation before switching to tracking. This step of the configuration process is part of the main *DroneResponse* system implementation and not further discussed in the implementation of the configuration tool.

As a result of these four steps, the user has a complete activity diagram summarizing the configured mission. All the configurations are nested in a JSON format object that is ready to be sent to the main DroneResponse system in order to launch and execute the mission.

Figure 16 summarizes the process the user go through in order to execute a mission

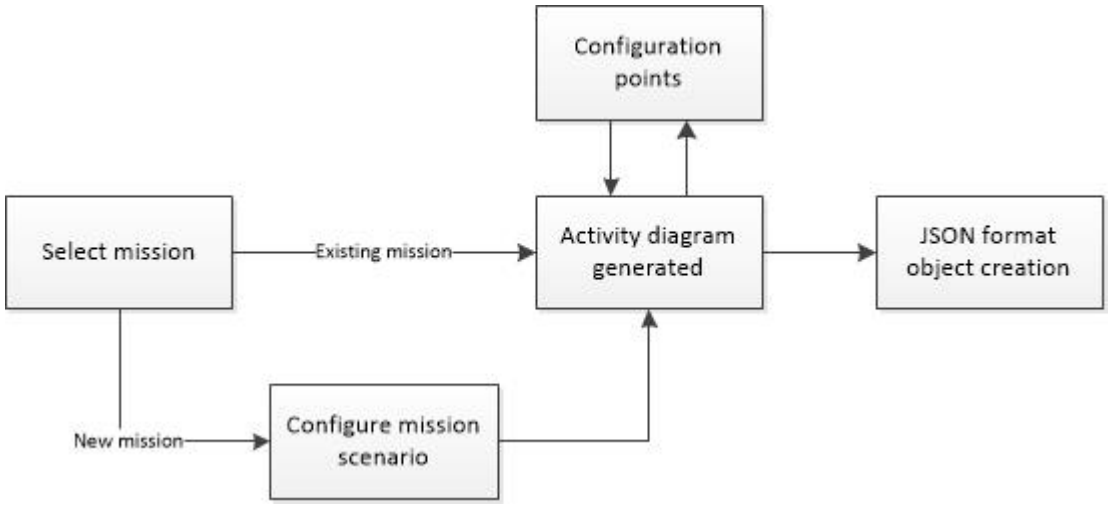


Figure 16: Steps of the configuration process using the configurator tool

The next subsection discusses implementation of the actual configurator.

5.3. Implementation of *DroneResponse* configurator tool

We use the *Angular*[5] framework to build the configurator tool. We decide to choose this framework because components of the main *DroneResponse* system are built using it, therefore easing communication with these components. *Angular* is a front-end framework to develop web applications using Typescript (javascript-based language including strong typing).

The configurator tool implementation is done using both merge feature model and activity diagram built in section 5.1 as well as their mapping to concrete component of *DroneResponse* architecture. It is important to note that although feature models offer a way of dealing with variability by combining various features into different products, we only allow configuration of missions that are compliant with the merged activity diagram (see 5.1.5) for now. This ensures safety in the definition of missions by preventing unsafe and untested interaction of some features. However, we consider to soften this constraint in the future.

We aim at designing an iPad-friendly application. Emergency responders have to quickly configure *DroneResponse*, maybe on the field. They therefore won't use a mouse and keyboard to input data, answer questions and so on. We start designing the configurator tool with a pretty simple landing page (see Appendices) which prints existing missions as well as a button to start a new configuration.

5.3.1. Data structure and questions rendering

Since we have a complete activity diagram that brings together all possible steps that might be used to configure a new missions, we have to identify places on the graph where choices are made. Then, we can define questions that explicit this choice points. These questions will be the ones asked to the user in order to draw the right path on the main activity diagram that represents their custom mission.

[5] https://angular.io/

We identify a set of 11 questions that covers the entire system-wide activity diagram. These questions are depicted in Figure 17.

ID	Question	Answers
Q1	What type of mission?	Fire fighting support, environmental sampling, Search, Delivery, Surveillance
Q2	How will you define flight paths?	Region, Waypoints
Q3	Are you fighting a structural fire or a ground fire?	Structural, Ground
Q4	What type of environment are you working in?	Water, Land, Ice, Snow
Q5	What are you surveying?	Flood, Traffic, Other
Q6	Are there independent rescue teams?	Yes, No
Q7	Should drones track the victim?	Yes, No
Q8	Will your mission deliver rescue equipment to the victim?	Yes, No
Q9	Do drones have on-board sample analysis capabilities?	Yes, No
Q10	What are you sampling ?	Water, Air, Radio
Q11	Are you searchin for a victim or a suspect ?	Victim, Suspect

Figure 17: Set of questions that covers all kind of missions that can be built

We notice that an answer to a question implies a series of sub-questions, therefore we decide on organizing the questions into a tree hierarchy. Because of the tree data structure, we can easily nest the questions and have proper organization of the data. We will then be able to iterate over the tree in order to ask the questions to the user. Figure 18 explains the hierarchy of questions. Q1 is asked to the user, according to the user's answer that might be "Fire", "Surveillance", "Sampling", "Search" or "Delivery", the questions located in the corresponding nodes are asked. For example, if the user inputs "Sampling" when asked Q1, he will then be asked Q2, Q10 and Q9. Because an answer to a question might bring several sub-questions (just like "Sampling" as in the previous example), we had to create intermediary nodes in our data structure that select all of the children. These intermediary nodes are drawn as black dots on Figure 18.

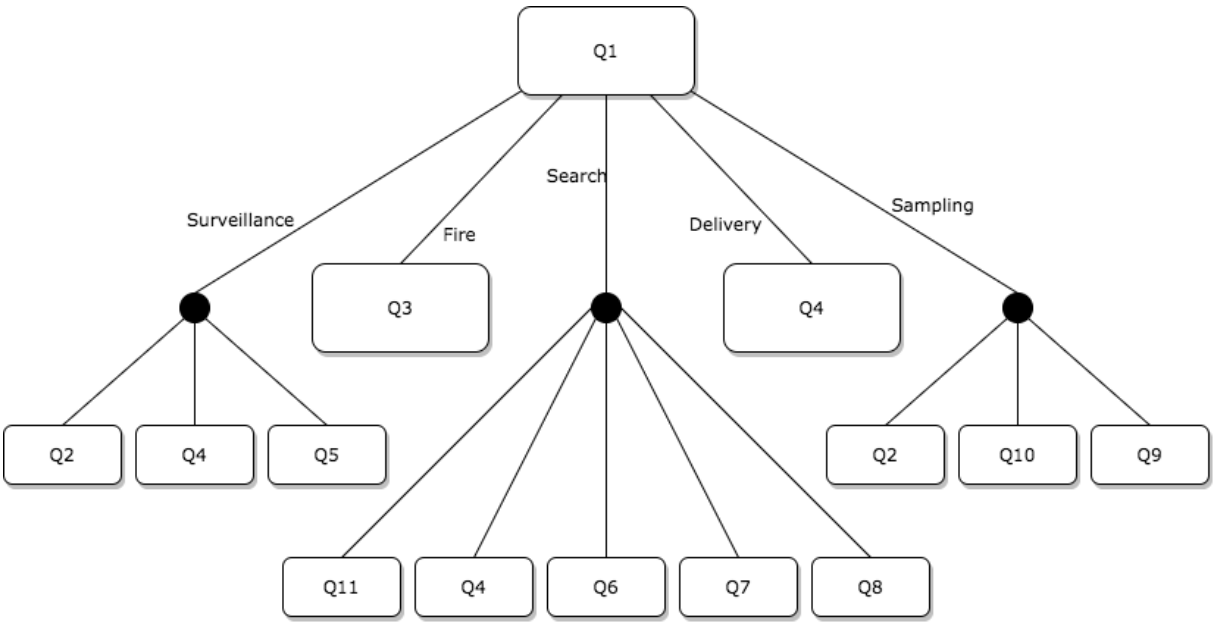


Figure 18: Data structure of the questions stored as a tree hierarchy

Our *Angular* project contains multiple components that interact together in order to achieve the desired goal. The main component of our architecture is the “Configurator Component”. This component handles the main logic for the configuration process.

Figure 19 shows our tool high-level architecture and relations between the main components.

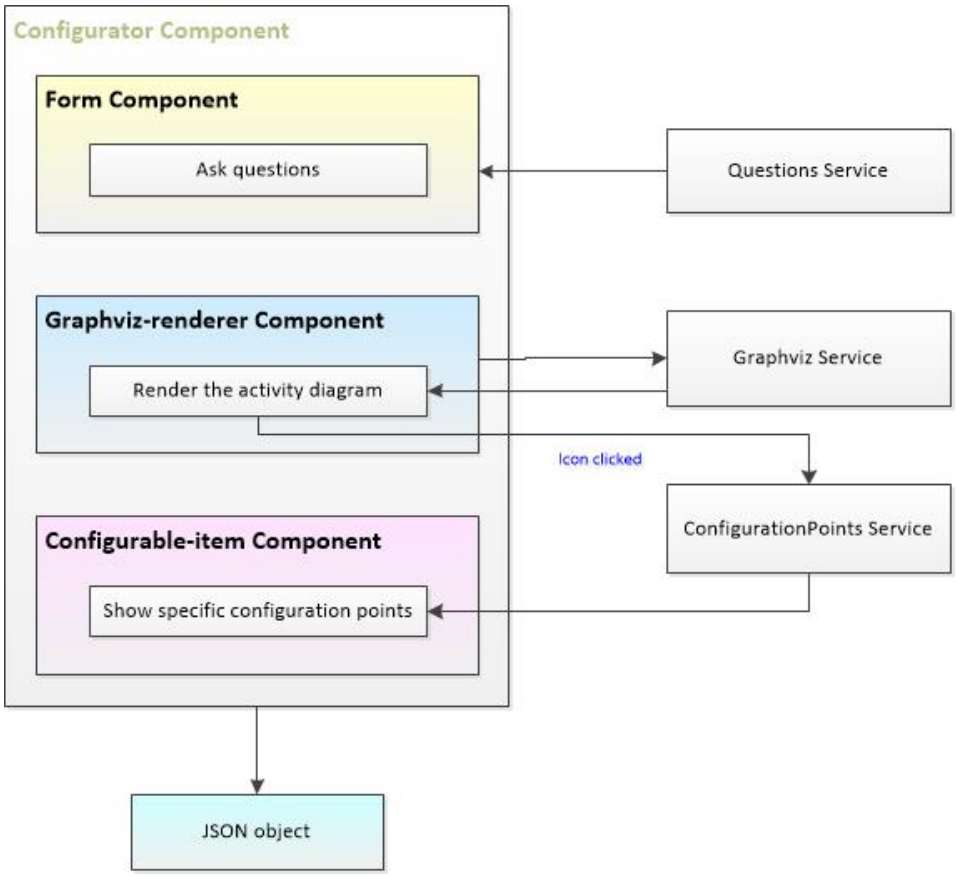


Figure 19: Configurator tool main components and services

It first retrieves the questions data structure stored in an *Angular* service and extracts the first node of the tree which is the parent node of all other nodes. The corresponding question (Q1) and answers of the node are then printed to the user. *Angular* then waits for the user to input an answer into the system. According to the answer the user picks, the code retrieves the corresponding child nodes and asks the next question on the user interface. Since *Angular* is a reactive framework, it updates the components itself every time a change is detected. We therefore only need to update the variables storing the current question and answers, and let *Angular* update them on the UI for the user. We use a simple breadth-first search algorithm to visit the question tree and retrieve the new nodes, therefore consuming the nodes closest to the root first. We choose this algorithm instead of a depth-first search algorithm because questions at a higher place in the hierarchy are more general than questions located lower in the hierarchy. Therefore, higher-located questions most of the time relate to core nodes of the graph while lower-located questions relate to less important or secondary nodes on the activity diagram.

This way, we have information about main nodes first which allows us the dynamically render the graph after each answer of the user. For example, we can see that Q2 and Q4 are the first questions asked if the user picks “Sampling”, “Delivery” or “Surveillance” mission. These two questions are respectively related to area and terrain definitions for the mission which are involved in the first nodes of the activity diagram while questions like Q5, Q8 or Q9 which respectively relate to survey type, extra delivery and on-board capabilities, are more specific to some mission type so they should be asked later.

We now need a way to transmit the user’s answers to another component of our app that will dynamically render the corresponding activity diagram. To do this, we create a TypeScript object that we name “MissionConfiguration”. This object holds 11 variables referring to our 11 questions. A code snippet representing the object is depicted below.

```
export class MissionConfiguration {
  mission: string
  terrain: string
  fireType?: boolean
  surveyType?: string
  sampleType?: string
  waypoints?: boolean
  isSuspect?: boolean
  extraDelivery?: boolean
  onboardAnalysis?: boolean
  independentRescueTeams?: boolean
  tracking?: boolean
}
```

An interrogation point in TypeScript means an optional variable. Therefore, we can see that only the mission and terrain types are mandatory to all missions. Our *Angular* code checks when getting the user answer to which variable of this object it refers to and stores the answer into it. The object’s content is updated every time the user answers a question. When no more questions remain, this process of visiting the question tree and building the “MissionConfiguration” object stops.

5.3.2. *Rendering a graph based on user inputs*

We decide on using *Graphviz*[6], an open source graph visualization software to build activity diagrams of the mission. *Graphviz* graphs are created using the DOT language. We used *d3-graphviz*[7], a JavaScript library to integrate *Graphviz* in *Angular*. We write two components that are relative to *Graphviz*, the "Graphviz-renderer Component" to render the graph in our app and the "Graphviz Service" to handle the logic for building the graph. The "MissionConfiguration" object is sent to the "Graphviz Service" after each user input. The service uses a set of if-else statements to check the defined variables in the object then build the DOT string of the graph. An example of the graph string built for a "River search-and-rescue" scenario is depicted at Figure 20. A graph is always created using a "digraph" statement. Then, information about the graph like color of the nodes, spacing, font family and so on are provided. Finally, nodes and transitions are declared. Nodes are basically declared using a name then square brackets in which we provide information about the node like the label, the shape or an image. Images are the icons we refer to in sub-section 5.3.3. Therefore, nodes containing an icon are nodes linked to a concrete component of the *DroneResponse* architecture. These are configuration points that the user may configure by clicking on the icon on the graph. Transitions are declared using an arrow inside two node names. We also provide information about the transition in square brackets.

[6] http://www.graphviz.org/
[7] https://github.com/magjac/d3-graphviz

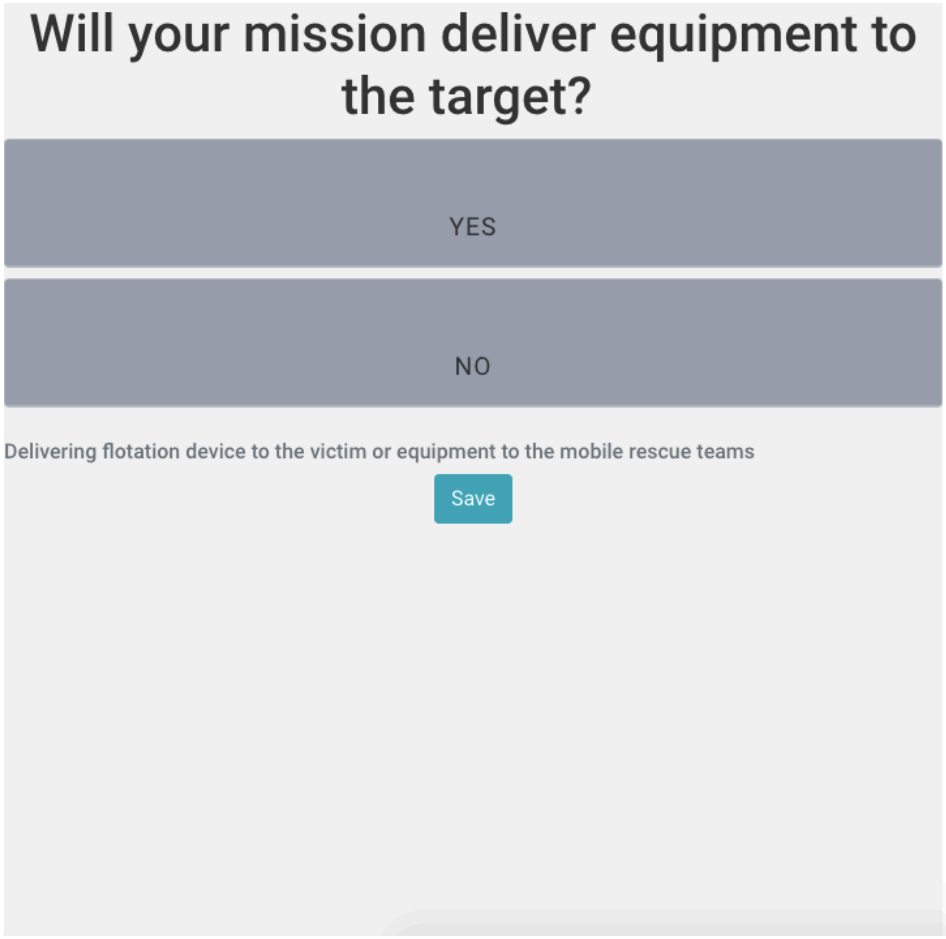
```
digraph G {
  graph [ranksep="0.4", pad=".5", nodesep="0.3"]
  edge [fontsize=12,fontname="times:italic"]
  node
[fontsize=16,fontname="times"shape=box,style=filled,imagepos="mr",margin="0.
4,0",fillcolor = grey94]//"/pastel18/2"]

  START_MISSION [label=" Start search mission",shape=ellipse]
  REGION [ label="Define Area and Generate flight routes",
image="../../assets/icons/route.png", color=red]
  FLY_TO_LOCATION [ label="Fly to defined area"]

  START_MISSION -> REGION [label=""]

  LAUNCH [ label="Connect and launch drones",
image="../../assets/icons/drone.png", color=red]

  REGION -> LAUNCH [label=""]
  LAUNCH -> FLY_TO_LOCATION [label=""]

  SCENE_RECONSTRUCTION [ label="Scene Reconstruction activated for snow"]

  FLY_TO_LOCATION -> SCENE_RECONSTRUCTION [label=""]

  SEARCH [ label="Search"]
  TARGET_FOUND [ label="Victim found"]

  SCENE_RECONSTRUCTION -> SEARCH [label=""]
  SEARCH -> TARGET_FOUND [label=""]

  VICTIM_RESCUED [ label="Victim rescued by human responders"]
  TRACK_TARGET [ label="Track victim"]

  TARGET_FOUND -> TRACK_TARGET [label=""]
  TRACK_TARGET -> VICTIM_RESCUED [label=""]

  DRONES_RECALLED [ label="Drones recalled by Incident commander"]

  VICTIM_RESCUED -> DRONES_RECALLED [label=""]
  DRONES_RECALLED -> RETURN_HOME [label=""]

  RETURN_HOME [ label="Return Home"]
  END_MISSION [label=" End mission",shape=ellipse]

  !RETURN_HOME -> END_MISSION [label=""]
}
```

Figure 20: Graph string built for a river search-and-rescue scenario

Once the graph string is built, it is sent to the other *Graphviz* component. The “Graphviz-renderer Component” sets options for the rendering. We use the following options for the rendering of our graphs:

- Transition: This option defines the transition use when new nodes are added to the current instance of the graph. We use an “easeLinear” transition.
- AddImage(path: String): *d3-graphviz* requires declaring images that are to be used in the graph into the renderer. We therefore add this option multiple times to add every icon we use.
- Width / Height: Sets the SVG graphs width and height.
- Fit: Boolean used to cause the graph size to fit to the SVG size previously defined. We set it to True for our graphs.
- Zoom: Boolean, when set to True, it allows the user to zoom in and out on the graph. We set it to False to avoid rescaling the graph which causes a bad rendering.

The component then just renders the graph on the UI. An example of the graph built for a “River Search-and-rescue” scenario is depicted at Figure 21.



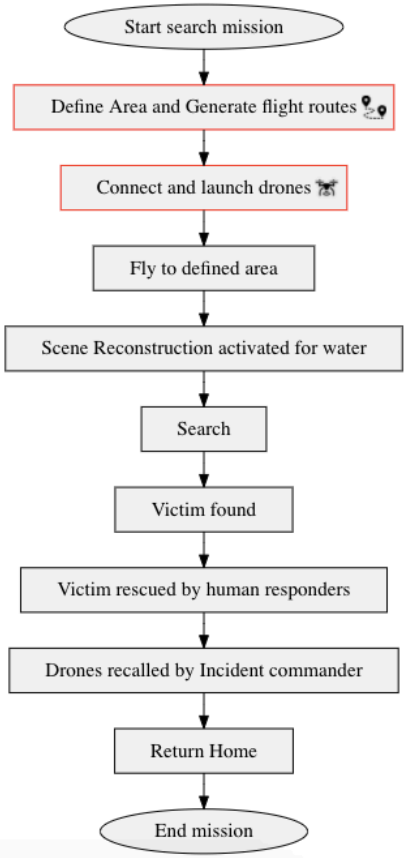


Figure 21: Screenshot taken during the configuration process for a river search-and-rescue scenario

5.3.3. Configuration points implementation

As discussed in section 5.2, configuration points are also part of the configuration process. Some concrete components of *DroneResponse* system can be configured directly from the rendered graph. These configurations are as previously described, at a lower-level than the questions answering/graph building processes that assemble the mission at a high-level.

We previously linked concrete component of the architecture to nodes of the activity diagram in the requirements section of this paper (see 5.1.6). We will now use icons on the graph to configure these components. We create a "ConfigurationPoints Service" in our *Angular* app that stores the data about the configurations points.

The data structure at this point looks like the following code snippet.

```
export class ConfigurationPoint {
  name: string;
  icon: string;
  questions: Question[];
  configured: boolean;
}

export class Question {
  question: string;
  answers: string[];
}
```

Data about configuration points are stored by declaring a variable whose type is an array of "ConfigurationPoint". Configuration points contain:

1. A name as an identifier.
2. An icon that will be displayed on the node corresponding to the concrete component it configures. Icons are unique, this means that an icon only refers to one configuration point.
3. A set of questions and answers to configure the component.
4. A Boolean storing the state of the configuration point. This variable is set to True if the user configured the configuration point and False if not.

In order for users to click on the graph icons, we add event listeners to each icon on the graph. Each time a user clicks an icon, an event is emitted, containing the icon name. The parent component which is the “Configurator Component” will handle the event and send the icon name to a new component, the “Configurable-Item Component”. This new component opens to the right of the screen, next to the graph. Using the name of the icon it received from the parent component, this component calls the “ConfigurationPoints Service” to retrieve the configuration point data linked to that icon. It then just prints the questions and possible answers. When the user submits his choices, the “Configurable-Item Component” closes. We use a colored border on the graph nodes that contain icons. An icon that has already been configured has a green border while an icon not configured is red. Figure 22 shows a user configuring a mission in which two concrete components can be configured on the graph. The first one has already been configured, therefore colored green and the second one has been clicked thus about to get configured.

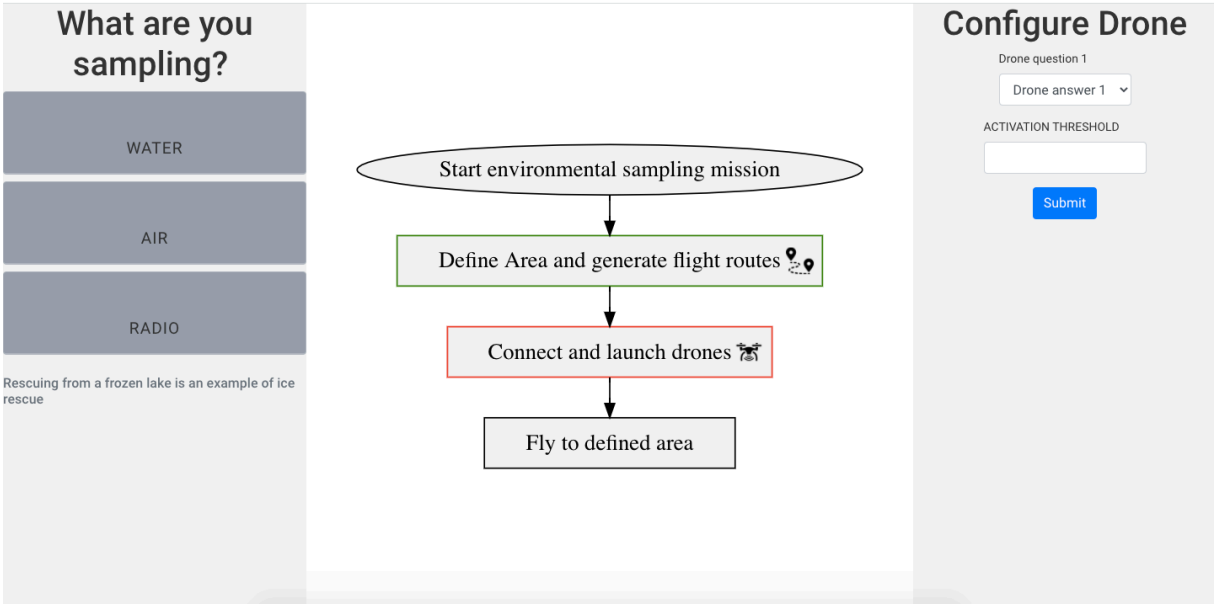


Figure 22: Screenshot taken during the configuration process in which the user configure a configuration point.

5.3.4. Creating a JSON format mission specification

As explained in the configuration process part (see 5.2), the output of all these configurations are of two types, (1) a visualization through an activity diagram of the mission configured and (2) a JSON format object holding the resulting configurations.

Just like the graph building, such an object can be created in advance and updated after every user's input. We decide on creating an object that stores:

a. Metadata about the mission which are the type ("Search", "Sampling", "Surveillance", "Delivery" or "Fire") and the terrain type ("Land", "Water", "Ice" or "Snow").
b. Activities of the mission which basically are the nodes of the activity graph. Just like in our *Graphviz* string building process, nodes have a name and a label. We store both in our object.
c. Transitions, these are composed of a starting, an end node and optionally a condition.

All of this information is actually available in the "Graphviz Service" which is in charge of building the graph string. We therefore add a few methods that push each new node or transition into the JSON formatted object. We also set the mission and terrain types in the object as soon as the user answers these questions. An example of the JSON object output at the end of the configuration of a delivery scenario is depicted in the following code snippet.

```
{"mission": [
  {
    "metadata": {
      "MissionType": "Delivery",
      "SubType": "LAND"
    },
    "activities": [
      {
        "nodeName": "START_MISSION",
        "Label": "Start delivery"
      },
      {
        "nodeName": "REGION",
        "Label": "Define target waypoints"
      },
      {
        "nodeName": "LAUNCH",
        "Label": "Connect and launch drones"
      },
      {
        "nodeName": "FLY_TO_WAYPOINT",
        "Label": "Fly to target waypoint"
      },
      {
        "nodeName": "FIND_DROPLOCATION",
        "Label": "Identify a safe drop spot"
      },
      {
        "nodeName": "DROP_SUPPLIES",
        "Label": "Deliver supplies"
      },
```

```
    {
      "nodeName": "FIND_RTL_LOCATION",
      "Label": "Identify a home-based within flying range"
    },
    {
      "nodeName": "RETURN_HOME",
      "Label": "Return Home"
    },
    {
      "nodeName": "END_MISSION",
      "Label": "End mission"
    }
  ],
  "transitions": [
    {
      "from": "START_MISSION",
      "to": "REGION"
    },
    {
      "from": "REGION",
      "to": "LAUNCH"
    },
    {
      "from": "LAUNCH",
      "to": "FLY_TO_WAYPOINT"
    },
    {
      "from": "FLY_TO_WAYPOINT",
      "to": "FIND_DROPLOCATION"
    },
    {
      "from": "FIND_DROPLOCATION",
      "to": "DROP_SUPPLIES"
    },
    {
      "from": "DROP_SUPPLIES",
      "to": "FIND_RTL_LOCATION",
      "condition": "Confirmed"
    },
    {
      "from": "FIND_DROPLOCATION",
      "to": "FIND_RTL_LOCATION",
      "condition": "Cancelled"
    },
    {
      "from": "FIND_RTL_LOCATION",
      "to": "RETURN_HOME"
    },
    {
      "from": "RETURN_HOME",
      "to": "END_MISSION",
    }
  ]
```

```
    }
  ]
}
```

The JSON object is then sent to the core *DroneResponse* system. It will impact the system in several ways including central control mechanisms, onboard autonomy, mobile units, user interface and so on. However, the way the system will interpret this configuration object in order to configure all the related components is way out of the scope of this document. We therefore limit our work to what has been described in this approach which was to create an object holding data about configuration of a mission and a visualization of the mission customized based on user inputs.

5.4. User study and approach evaluation

The configurator tool discussed in the previous section (see 5.3) is at this point working as specified. In this section, we evaluate our approach and organize a small user study.

Our approach is evaluated with two sets of questions that this section should answer:

a) Is the configurator tool able to generate valid mission specifications for the 7 initial use cases from Figure 10? Also for new additional use cases?
b) Can external users use our configurator tool to generate valid mission specifications of emergency response scenarios? What are the main challenges they encountered?

Question a: We first of all start by doing a set of validation tests using the configurator. We configure mission specifications for each of the initial 7 use cases used in the requirements engineering process of this approach (Figure 10). Although some nodes were placed at the wrong place or transitions were missing from a node to another, most of the graphs built were as expected. We corrected a few mistakes in the code to fix these minor issues.

Since all the initial scenarios were built successfully using the configurator, we now aim to generate previously unknown mission scenarios of emergency response using drones in order to assess our configurator's ability to create custom missions. In order to do that, we wrote 13 more use cases based on resources found on the internet. These new use cases are detailed in Figure 23.

Use-case ID	Use-case name	Use-case ID	Use-case name
UC8	Chemical spill	UC15	Flood support
UC9	Avalanche rescue	UC16	Earthquake damage
UC10	Suspect tracking	UC17	Rip current rescue
UC11	School shooting	UC18	Lost kayaker
UC12	Radiation detection	UC19	Volcanic eruption
UC13	Man overboard	UC20	Utility inspection
UC14	Crowd control		

Figure 23: Additional use cases of mission scenario that DroneResponse should handle

We repeat the test of configuring each of the scenarios using the configurator tool. The results are a bit less satisfying than the ones for our 7 initial use cases. Most of the mistakes we identified are syntax and vocabulary issues. For example, our "victim tracking" node is not adapted for UC11 which is a school shooting. "Suspect tracking" would be a better terminology for this particular mission. However, since the core tasks of missions are shared across the scenarios, graphs generated are not that bad if we correct terminology issues. This might be done asking a few more specific questions.

Question b: We then switch to the second set of question to evaluate this approach. For this evaluation, we need some external users that have never experienced our configurator tool but that have previous experience with UAVs. We therefore recruit 5 participants to participate to our short user story. This one is done during an online video call and lasts in average 30 minutes per participant.

Each of the participant is introduced to our *DroneResponse* project with an online short video that explains how we envision using drones to support emergency responders in time-critical tasks. We then introduce them our configurator tool and use it to configure UC1 "River search-and-rescue" for them to understand the process of configuring missions. Each participant is assigned 2 mission scenarios to configure plus one they can choose themselves in our list of use cases depicted at Figure 23. Participants are asked to describe in a verbose way what is on their mind during the entire configuration process so that we can really get a lot of details about the configuration process.

Once they are done configuring their 3 missions, each user is asked if he could correctly configure the assigned missions. Two out of the five participants find the questions accurate and correct while the three others would have liked more questions to get more precise graphs of the missions. As an example, we ask if emergency responders require a delivery in case of river search-and-rescue but we might provide various supplies and therefore add a question asking this to the user configuring the mission.

Then, every participant agreed on the usefulness of the visualization part of the configurator tool. They all said it allows them to fully understand the mission they just configured. Participants were finally asked about suggestions to improve our app. A few participants pointed out that terminologies used in the questions and nodes of the graph are not enough context-specific. They said it might be better to involve emergency responders directly in the creation of questions and vocabulary for the missions. As a conclusion, this approach has a lot of potential, especially because of the visualization side of showing missions.

PART IV. Conclusions and perspectives

Chapter 6. Work summary

In the contribution part of this thesis, we came with two approaches to quickly eliciting requirement for mission scenarios in order to configure our *DroneResponse* system accordingly. Both approaches use traditional requirement engineering processes in order to build the system as a product line. Product lines are useful to describe variability in products that have commonalities and specificities. Variability has been documented using textual use cases, activity diagrams and feature models. Even though the processes for creating the product line have been different for our two approaches, both started from a set of well-defined use cases and have been merged into a product line.

Approaches also differs in the way of deriving a new product when the product line has been created. The first approach created a static mapping from a set of tags defined in the requirement engineering process to features of the system-wide feature model of *DroneResponse*. A classifier has been trained in order to map a previously unknown use case scenario to the feature model, therefore ending up with a configuration of the system. The test use case is a verbal description of a mission scenario, allowing emergency responders to **quickly elicit requirements** for a new mission. This can also be useful to pretty fast update the mission goals during a mission if events changing the mission occur. Emergency responders would then just describe changes and let the system treat the new request.

The second approach provided a tool to configure new missions through a basic user interface. The user interface asks a series of question to the user that relates to our underlying data structure. An activity diagram representing the user's choice is then rendered after each user's input. This approach gives a high-level view of what the mission that is to happen looks. Users can also configure lower-level settings related to concrete components of *DroneResponse* architecture. When the user ends up the configuration process, a JSON format object holding all the configurations is sent to the main system.

Our work has several threats to validity. First of all, the configurations with output at the end of our processes are pretty high-level, we therefore do not discuss how these configurations will be used in the main *DroneResponse* system. Secondly, we should have included domain expert in our user studies. Indeed, users that configured the system for the studies are drone flyers but not emergency responders working in time-critical mission scenarios. Therefore, including more related expert would allow us to improve our understanding of the requirement on how to build our *DroneResponse* system.

Chapter 7. Future work

One of the main downside of the first approach is that we trained our classifier to tag pretty short sentences that only carry one meaning or "concept". However, the verbal description of missions come as a full block of text. We therefore needed to split the full text into sentence-like chunks. This problem should be addressed as future work by training a multi-classifier that would handle several tags for one use-case step. The classifier could then tag a full block of text without needing to split it into small steps.

For our second approach, we aim in future work to also use the activity diagram generated during the configuration process to show the situation of each drones during a mission. This would allow emergency responders to know what tasks are performed by which UAVs during a mission. An important challenge that still has to be address is including the configuration points into the JSON format object output at the end of the configuration process. The current version of the code is only storing metadata and activities about the overall mission but does not include the configuration points yet. We also refine our data structure in order to have a much stronger relation between the graph creation and the questions. For now, our algorithm visits a tree of questions, asks these questions to the user and build an object storing the users answer before creating the graph. Work is progress in actually storing questions and activity nodes inside one Angular service. Nodes are created in the graph if conditions attached to them (which are the questions) are fulfilled. However, the timeline did not allow the author to include this version of the code in the thesis.

Finally, as described in section 5.4, the user study is planned to be improved by including real domain experts and have a better feedback of our system and requirements to improve *DroneResponse*.

Chapter 8. Conclusion

Both approaches show great potential but also requires way more work in order to be successful. This has been described in the future work part of this chapter. The paper written during the first approach will probably be submitted when changes are done. However, the second approach has been the main part of the internship work for the author. We spent most of the four months working on this second approach which ended up with a submitted an accepted paper for the software product line conference 2020 (Cleland-Huang, et al., 2020).

PART V. Bibliography

Agrawal Ankit [et al.] The Next Generation of Human-Drone Partnerships: Co-Designing an Emergency Response System [Revue] // CHI'20. - Honolulu, HI, USA : [s.n.], 25-30 April 2020.

Arias Darwin Armando Mora [et al.] Unmanned Aerial Vehicle for Rescue and Triage [Revue] // Botto-Tobar M., Zambrano Vizuete M., Torres-Carrión P., Montes León S., Pizarro Vásquez G., Durakovic B. (eds) Applied Technologies. ICAT. Communications in Computer and Information Science.. - Cham : Springer, 2019. - Vol. 1194.

Bühne Stan [et al.] Scenario-based application requirements engineering [Conférence] // Software Product Lines. - [s.l.] : Springer, 2006. - pp. 161-194.

Baresi Luciano Activity Diagrams [Section] // LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. - 2009.

Bragança Alexandre et Machado Ricardo Automating Mappings between Use Case Diagrams and Feature Models for Software Product Lines [Conférence] // 11th International Software Product Line Conference (SPLC). - 2007.

Cleland-Huang Jane [et al.] Requirements-Driven Configuration of Emergency Response Missions with Small Aerial Vehicles [Conférence] // Software Product Line Conference'20. - Montreal, Canada : [s.n.], 2020.

Cleland-Huang Jane, Vierhauser Michael et Bayley Sean Dronology: An Incubator for Cyber-Physical Systems Research [Conférence] // IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER). - 2018.

Cockburn Alistair Writing Effective Use Cases [Ouvrage]. - [s.l.] : Addison-Wesley Longman Publishing Co., Inc.75 Arlington Street, Suite 300 Boston, MAUnited States, 2000. - p. 304.

Favaro John, Griss Martin et d'Alessandro Massimo Integrating feature modeling with the RSEB [Conférence] // Proceedings. Fifth International Conference on Software Reuse (Cat. No.98TB100203), Victoria, BC, Canada. - 1998. - pp. 76-85.

Fleck Mathias Usability of Lightweight Defibrillators for UAV Delivery [Conférence] // CHI EA '16: Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems. - 2016. - pp. 3056-3061.

Guttierez Javier [et al.] Visualization of Use Cases through Automatically Generated Activity Diagrams [Revue] // Czarnecki K., Ober I., Bruel JM., Uhl A., Völter M. (eds) Model Driven Engineering Languages and Systems. Lecture Notes in Computer Science. - Berlin : Springer-Verlag, 2008. - Vol. 5301.

Jacobson Ivar Use cases - Yesterday, today, and tomorrow [Revue] // Software and Systems Modeling (SOFTW SYST MODEL). - 2004. - pp. 210-220.

Kang Kyo [et al.] Feature-Oriented Domain Analysis (FODA) Feasibility Study [Rapport] / Softare Engineering Institute. - 1990.

Pohl Klaus, Günter Böckle et Van der Linden Frank Software Product Line Engineering: Foundations, Principles and Techniques [Ouvrage]. - Secaucus : [s.n.], 2005. - pp. 4-18.

Pudlitz Florian, Brokhausen Florian et Vogelsang Andreas Extraction of System States from Natural Language Requirements [Conférence] // International Requirements Engineering Conference (RE). - [s.l.] : IEEE, 2019. - pp. 211-222.

Turner Ian, Harley Mitchell et Drummond Christopher UAVs for coastal surveying [Revue] // Coastal Engineering. - 2016. - Vol. 114. - pp. 19-24.

Yue Tao, Briand Lionel et Labiche Yvan An Automated Approach to Transform Use Cases into Activity Diagrams [Revue] // Kühne T., Selic B., Gervais MP., Terrier F. (eds) Modelling Foundations and Applications. Lecture Notes in Computer Science. - Berlin : Springer, 2010. - Vol. 6138.

PART VI. Appendices

Appendice 1: Manually tagged use-case for creating and updating a heatmap of an on-fire building

Use Case: Creating and updating a heatmap of an on-fire building

Description

Several drones are dispatched to examine a building that is on fire and create a heatmap of it that will be updated every X minutes (X provided by the user).

Primary Actor

Multiple UAVs

Supporting Actors

Firefighters

Main Success Scenario

1.A 911 call is received for a fire at a specific address. [emergency_call]
2.The operator dispatches fire trucks to the scene equipped with drones. [drones_transported_to_site]
3.DroneResponse downloads current NOTAMs including no-fly zones. [flight_prohibition]
4.Upon arrival at the fire site, the drone operator activates the drones. [drones_activated]
5.The drones activate their cameras and start streaming video. [camera_activation]
6.The drones plan coordinated flight routes in order to efficiently cover the building. [routes_planning]
7.The drones take off. [take_off]
8.Each drone uses its onboard imagery to avoid obstacles (walls, chimneys) and to fly around the building. [collision_avoidance]
9.Each drone sends imagery to a central server. [image_stream]
10.The DroneResponse central server analyzes the imagery and generates a live heatmap of the building. [scene_reconstruction]
11.The DroneResponse central server analyzes the imagery and reconstructs a 3D view of the scene. [scene_reconstruction]
12.Dynamically generated scenes are regenerated every period. [scene_reconstruction]
13.Whilst mapping out the building, a drone detects a victim in the window. [target_detection]
14.The drone hovers in place by the window in which the victim is found. [target_tracking]
15.The drone sends a notification to the incident commander and streams video of the victim. [d2h_event_notification]
When notified by the drone operator, the drone flies to the landing site and lands. [rtl]

Appendice 2: DroneResponse landing page. Existing missions are on the top-right of the screen.

Use Case: Search and find a victim in area of river
ID: UC-1

Description
Multiple UAVs dispatched to search for victim in river

Primary Actor
Drone Commander

Supporting Actors
Semi-autonomous UAV

Stakeholders and Interests
Fire department engaged in river rescue
FAA concerned with flight regulations
General public

Pre-Conditions

- Dronology system is active
- Multiple UAVs are equipped with cameras and are placed on the yyund and are activated
- Firefighters have marked area of river to be searched
- Search plan has been generated
- DroneResponse is running and UAVs are displayed on map
- A victim is in the search area
- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition
The victim is found by a UAV and actively tracked until a first responder takes over the rescue operation
Failure end condition:
The victim is not found or the victim is found but not actively tracked.

Trigger
The Drone Commander activates the search.

Main Success Scenario

1. DroneResponse continually tracks and displays the location and state of each UAV.
2. The UAVs takeoff.
3. The UAVs commence their individually assigned search patterns.
4. After arriving at the scene, each drone activates its camera.
5. Each UAV processes imagery from its camera using a trained river-victim image detector.
6. One UAV (Finder) flies over the victim and its onboard image recognition software detects the victim with confidence greater than a predefined threshold.
7. The UAV raises an alert.
8. The UAVs switches to `active_tracking' mode.
9. DroneResponses raises an alert and asks for human confirmation that the victim found by the drone.
10. The Drone Commander confirms `active tracking'.
11. The Drone Commander notifies the Incident Commander who confirms the sighting and directs human responders in their boat to rescue the victim.
12. The Drone Commander confirms that the Finder-UAV has sufficient battery to continue active-tracking.
13. He/she recalls all other drones to their home-base.
14. Human responders arrive at the scene.
15. The Drone Commander recalls the Finder-UAV to its home-base.

Exceptions

1a. In step 1, communication is lost with an individual drone (Human-Drone)
  1a.1 A warning message is displayed depicting the duration of time for which communication has been lost

2a. In step 2, one of the UAVs fails to takeoff or has to be recalled due to mechanical failure during flight.
  2a.1 If an alternate UAV is prepped for flight, that UAV is dispatched in place of the failed UAV.
  2a.2 If no alternate UAV is prepped for flight:
    2a.2.1 The search paths are re-generated based on the reduced number of available UAVs.
    2a.2.2 The adjusted mission plans are sent to each UAV in flight.
    2a.2.3 Each UAV proceeds to execute its new mission plan.

4. In step 4, the UAV detects a possible victim at a confidence level below the predefined limit but above the lowest `ignore' level.
   4a.1 The UAV raises an alert
   4a.2 DroneResponse saves the GPS coordinates of the sighting
   4a.3 The UAV continues its currently assigned route.
   4a.4 The Drone Commander reviews the streamed imagery
   4a.5 The Drone Commander confirms that the sighting is not a victim.

   In step 4a.5, the Drone Commander reviews the streamed imagery and is unable to reject the image as a false-positive sighting.
      4a.5.1 The Drone Commander requests additional imagery from the area around the sighting.
      4a.5.2 DroneResponse assigns a UAV to fly to the vicinity of the coordinates and to acquire additional imagery.

5. In step 5, the drone fails to provide victim's imagery and remains in "active-tracking" mode without Drone Commander's assertion. (Drone-Human).
   5a.1 If the drone does not receive any acknowledgement from the Drone Commander within a specified time, then it tries to route the images through other participating drones.
   5a.2 If (5a.1) also fails, drone flashes a red light indicating that it needs immediate human action to regain the communication.

5. In step 7, the communication between Drone Commander and incident commander fails.(Human-Human)

Use Case: Deliver defibrillator to a specific location
ID: UC-2

Description
When a victim is at a place that is hard to reach in a short time,UAV(s) dispatched to a specific location to send urgent resources.

Primary Actor
Receivers/UAV

Supporting Actors
Medic crew

Stakeholders and Interests
Medical professions engaged in first rescue materials

Pre-Conditions
- Dronology system is active
- Multiple UAVs are equipped with cameras and are placed on the ground and are activated
- Drones are able to carry limited weight
- There are enough drones to take care of the delivery process
- DroneResponse is running and UAVs are displayed on map
- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition
The medical package has been successfully collected by the target.
Failure end condition:
The medical package has not been collected by the target.

Trigger
The Drone Commander activates the delivery.

Main Success Scenario

1. A user calls 911 to report a medical emergency.
2. The operator identifies the location and uses Google maps to identify GPS coordinates for the delivery.
3. A ready-to-fly drone with a pre-attached defibrillator is selected.
4. The delivery drone downloads current NOTAMs including no-fly zones.
5. The delivery drone plans its route based on the prohibited areas.
6. The delivery drone takes off to a cruising altitude.

7. The delivery drone constantly checks for changes in terrain using the google map service and its own onboard sensors.
8. The delivery drone navigates the terrain autonomously, changing altitude to avoid hills etc.
9. The delivery drone arrives at the specified location.
10. Drones turn on their on-board cameras
11. The delivery drone uses onboard vision to identify a place to drop the package.
12. The drone requests permission for the drop from the operator.
13. The drone drops the package.
14. The receivers collect the needed package.
15. The drone requests a safe landing site from the operator.
16. The drone flies to the landing site and lands.

Exceptions

2a. Coordinate identification using Google maps failed
2a.1 Another mapping software is used to identify exact coordinates from the address
7a. The delivery drone fails to take off
7a.1 DroneResponse assigns the current drone configuration to a new drone that has a pre-attached defibrillator
7a.2 The new drone takes off to replace the one that failed
10a. For some reason, the delivery location has changed.
10a.1 DroneResponse identify the coordinates of the new location using Google maps
10a.2 DroneResponse transmits the new coordinates to the Drone
10a.3 The drone adapts its route and flies toward the new location
12a. The Drone Commander denies permission for the drop because the spot is not safe
12a.1 The Drone Commander uses the images from the drone camera displayed on DroneResponse to mark a safe spot for the drop on the map
12a.2 DroneResponse identify the coordinates for the spot marked on the map
12a.3 DroneResponse transmits the new coordinates to the drone
12a.4 The drones reaches the new location and drops the package

---

Use Case: Provide surveillance and information about a traffic accident
ID: UC-3

Description
One or more UAVs are dispatched to a traffic accident scene in order to provide information to the emergency responders.

Primary Actor
UAV's

Supporting Actors
Firefighters, Police, Medical crews

Stakeholders and Interests
Having specific information about a car accident (how many cars are involved, what is the exact location, traffic jams, people injured, etc) can help emergency responders having the proper reactions.

Pre-Conditions
- Dronology system is active
- Multiple UAVs are equipped with cameras and are placed on the ground and are activated
- DroneResponse received a approximative target position
- Search plan has been generated
- DroneResponse is running and UAVs are displayed on map
- An accident really is confirmed to be near the approximative target position
- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition
The accident has been precisely located and relevant information has been transmitted to emergency responders.
Failure end condition:
The accident has not been found ("Prank ?").

Trigger
The incident commander after receiving a 911 call

Main Success Scenario

1. A 911 call happens, the operator asks for the location, evaluates the situation and assigns a specific number of drones for the mission. (like how bad it is and where.)
2. The drones plan their routes to avoid buildings and other obstacles.
3. The drones takeoff.
4. The drones fly to the location of the accident.
5. When either one of the drones reaches the targeted area, it switches to "locate_incident" mode where it processes images from its camera in order to search for the accident.
6. Once one of the drones detects accident location.
7. It computes the GPS coordinates and sends the information to the operator.
8. The operator confirms that the accident site has been identified.
9. DroneResponse sends accident coordinates to other UAV's participating in the mission.
10. The UAVs all determine positions from which to observe the accident.
11. When a UAV reaches its targeted position it switches to "information_gathering" mode where the UAV streams imagery which is displayed on DroneResponse UI.
12. DroneResponse figures out the exact location.
13. The precise address of the accident and the surroundings of the road are sent to the emergency responders.
14. The time since the accident happened is displayed on DroneResponse monitor (time since 911 call?).
15. DroneResponse detects details such as number of cars, presence of fire, victims on the ground, and reports this in the GUI.
16. Specific information about the environment of the incident is gathered using specific detection algorithms (Traffic jam, Possible fire, Toxic leak, Available helicopter spot near the location, etc).
17. Specific information gathered by UAV's is processed and classified into specific emergency categories.
18. DroneResponse sends the information to the related emergency services. (Traffic information sent to the police, accident structure information sent to the firefighters, human related information sent to medical staff).
19. DroneResponse provides UAV's coordinates for a safe landing spot.
20. UAV's fly to the safe spot and land there.

Exceptions

3a. One drone fails to take off
3a.1 DroneResponse transmits the parameters from that specific drone to a new one
3a.2 The new drone takes off

6a. No drone can locate the accident.
6a.1 An alert is sent through DroneResponse claiming that the accident can't be located
6a.2 DroneResponse displays images from the camera of each drones
6a.3 The Drone Commander checks for the images and marks the accident as soon as it locates the accident on one drone's camera.

Use Case: Detecting radiation on a specific height
ID: UC-13

Description
Multiple UAVs take off to specific areas to detect radiation on a certain height.

Primary Actor
Drone Commander

Supporting Actors
Semi-autonomous UAV

Stakeholders and Interests
Firefighters

Pre-Conditions
Dronology system is active
Multiple UAVs are equipped with cameras and are placed on the ground and are activated
Firefighters have marked specific area and a height to be record
Recording_ route has been generated
DroneResponse is running and UAVs are displayed on map
All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition
Each UAV has successfully returned back the record of radiation to form a radiation map
Failure end condition:
Half of the UAVs are not able to send back records so that the radiation map is not precise enough to provide radiation information

Trigger
The Drone Commander activates the recording mission.

Main Success Scenario

1. The operator marks the incident area on the screen. [Scene_Annotation]
2. The operator launches DroneResponse and activates the "radiation_detection" mission. [Drone Activated]
3. The Drone Commander creates recording-routes for each drone and input a height for them to begin recording. [Scene_Annotation]
4. According to the numbers of routes, DroneResponse assigns each recording route for each drone.
5. Drones take off. [Take-off]
6. Drones follow the assigned routes. [Fly-to-Location]
7. Once each drone arrives at the first assigned coordinates, it switches to recording_mode (activates the radiation sensor). [Environment Sampling]
8. Each sensor records each coordinate in its route and labels the coordinates to its related zone based on thresholds using (mSV/Hour) as unit. [Environment Sampling]

Use Case: Creating and updating a heatmap of an on-fire building
ID: UC-7

Description
Several drones are dispatched to examine a building that is on fire and create a heatmap of it that will be updated every X minutes (X provided by the user).

Primary Actor
Multiple UAVs

Supporting Actors
Firefighters

Stakeholders and Interests

Pre-Conditions

- DroneX system is active
- Multiple UAVs are equipped with cameras and are placed on the ground and are activated
- Drones are set to stay at least fixed distance away from fire
- DroneX is running and UAVs are displayed on map
- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition
The fire heat map and other zone areas are successfully created for the firefighters to plan their heroic action.
Failure end condition:
The heat map is not created or not updated each _updatedFrequency_ seconds.

Trigger
The Drone Commander activates the scan mission.

Main Success Scenario

1. A 911 call is received for a fire at a specific address. [emergency_call]
2. The operator dispatches fire trucks to the scene equipped with drones.[drones_transported_to_site]
3. DroneResponse downloads current NOTAMs including no-fly zones.[flight_prohibition]
4. Upon arrival at the fire site, the drone operator activates the drones.[drones_activated]
5. The drones activate their cameras and start streaming video.[camera_activation]
6. The drones plan coordinated flight routes in order to efficiently cover the building.[routes_planning]
7. The drones take off.[take_off]
8. Each drone uses its onboard imagery to avoid obstacles (walls, chimneys) and to fly around the building.[collision_avoidance]
9. Each drone sends imagery to a central server.[image_streaming]
10. The DroneResponse central server analyzes the imagery and generates a live heatmap of the building. [scene_reconstruction]
11. The DroneResponse central server analyzes the imagery and reconstructs a 3D view of the scene.[scene_reconstruction]
12. Dynamically generated scenes are regenerated every _period_[scene_reconstruction]
13. Whilst mapping out the building, a drone detects a victim in the window.[target_detection]
14. The drone hovers in place by the window in which the victim is found.[target_tracking]
15. The drone sends a notification to the incident commander and streams video of the victim. [d2h_event_notification]
16. When notified by the drone operator, the drone flies to the landing site and lands.[rtl]

Exceptions

4.1: A drone is launched from the roof of the truck en-route to the scene. [drones activated]
4.2: The drone flies to the scene of the incident [fly-to location]
6.1: A subset of drones are assigned to map out the roof of the building. [Environment monitoring]
6.2: A subset of drones are assigned to map boundaries of the building [Environment monitoring]

At any time: A drone that is running low on battery returns home for a replacement battery. [battery replacement]
At any time: DroneResponse staggers battery replacements by bringing some drones in before the battery becomes repleted. [battery replacement]

6.3: An operator requests that a drone fly to a fixed location and stream video. [fly-to location]
X.1 The drone's onboard sensor shows that it has flown into a turbulent heat area [environment monitoring]
X.2, The drone adjusts its altitude and coordinates to avoid turbulent air. [flight adaptation]

Use Case: Avalanche
ID: UC-5

Description
Multiple UAVs are dispatched in order to look for people trapped in an avalanche.
Primary Actor
Several UAVs

Supporting Actors
Firefighters

Stakeholders and Interests
Firefighters/Medical Crew

Pre-Conditions

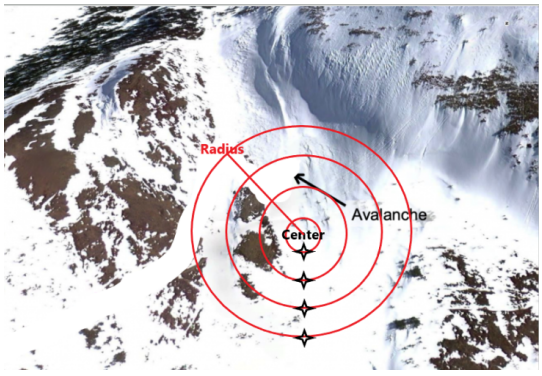
- Dronology system is active
- Multiple UAVs are equipped with thermal cameras and are placed on the ground and are activated
- Firefighters have marked the mountain area of to be searched
- Search plan has been generated
- DroneResponse is running and UAVs are displayed on map
- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition
The entire mountain area has been covered with thermal camera drones.
Failure end condition:
The mountain area defined has not been covered fully.

Trigger
The Drone Commander activates the search.

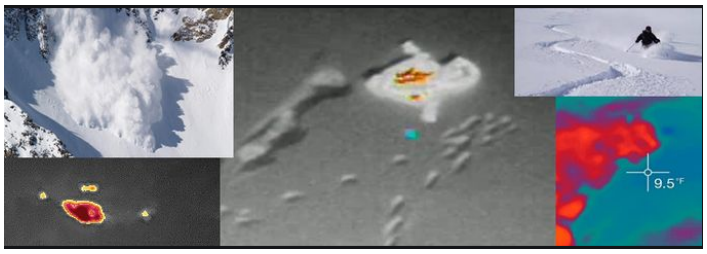
Main Success Scenario

1. An emergency call is received reporting an avalanche.
2. The operator launches DroneResponse and activates the "mountain_rescue" mission.
3. The Drone Commander specifies center coordinates and the size of a radius in order to fully cover the avalanche.



4. According to the size of the radius, DroneResponse selects how many drones are required for that mission and assigns a search plan for each drone. (As shown on the picture, each drone is covering his part flying as a circle).
5. DroneResponse assigns search areas for each drone for the mission.
6. DroneResponse creates routes to the coordinates for drones to avoid prohibited areas.
7. Drones take off and follow the assigned routes.

8. Drones heading to the assigned routes.
9. Each drone arrives at its associated coordinates and switches to “search_mode”.
10. Drones start using their thermal camera to look for trapped people in the snow.



11. One drone detects possible human-like heat sources(by setting a predefined temperature threshold for example).
12. It sends alerts back to firefighters through DroneResponse.
13. The Drone Commander views the imagery and confirms the existence of humans there.
14. This specific place is marked on DroneResponse.
15. Its coordinates are sent to the medical crew/firefighters.
16. Medical crew/firefighters have been assigned to head to the update coordinates.
17. Medical crew/firefighters arrive at the scene.
18. The drone Commander recalls the Finder-UAV to its home-base.

Exceptions

4a. The avalanche expands itself
- 4a.1 DroneResponse asks the Drone Commander to specify new center coordinates and a radius
- 4a.2 DroneResponse checks whether additional drones are required to cover the new surface
- 4a.3 DroneResponse assigns search areas for each drone of the mission

7a. One drone fails to take off
- 7a.1 DroneResponse assigns the parameters of the drone to a new drone
- 7a.2 The new drone takes off

8a. Unable to function properly because of external factors such as strong winds or heavy snow.
9a. The thermal camera lens is blocked by the snow so that it cannot perform well on thermal-detection.
10a. One drone detects heat sources close to, but under the predefined temperature threshold
- 10a.1 DroneResponse request additional imagery

15a. Medical Crew/Firefighters are unable to get to the coordinates due to external factors such as rift/valley or extreme weather
11/12/13/14/16. Bad/Lost connection

Use Case: Tracking a suspect running away from a shooting
ID: UC-6

Description
Multiple UAV’s are dispatched in order to identify and track a suspect in a shooting

Primary Actor
Drone Commander

Supporting Actors
Police officers

Stakeholders and Interests
Police officers are likely to require help tracking a suspect that tries to run off a crime scene

Pre-Conditions
- Dronology system is active
- Multiple UAVs are equipped with cameras and are placed on the ground and are activated
- Police officer specified the place the shooting happened
- Police officers are able to identify suspect for the drones to track
- DroneResponse is running and UAVs are displayed on map
- A suspect is in the search area

- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition
The suspect has been identified and the drones are able to keep track of him
Failure end condition:
The suspect has not been identified or the drones lost track of the suspect

Trigger
The Drone Commander activates the search.

Main Success Scenario

1. A 911 call is received reporting a shooting incident.
2. Police officers are deployed to the scene of a shooting with a set of drones.
3. Police officers place drones on the ground and activate them.
4. UAVs takeoff.
5. UAVs start surveilling the scene.
6. Drones stream real-time images from their cameras DroneResponse's UI.
7. An operator identifies the suspect on the screen and marks him.
8. The drone identifies a person matching characteristics of the shooter (e.g., carrying weapons, wearing a red jacket).
9. The drone switches to "suspect_tracking" mode.
10. Police officers surround the victim.
11. The drone hovers in the air and continues to stream video.
12. The suspect is arrested.
13. The drone flies to the landing site and lands.

Exceptions (Sorry numbers got messed up JCH)

1. The call includes a description of the shooter including his clothing. [Emergency Call]
2. DroneResponse failed to track the position of the police officers (Out of scope for this use case??)
    a. The incident commander assigns itself the right coordinates
3. Communication is lost with the drone [lost communication]
    a. A second UAV takesoff and replaces the missing drone [drone replacement]
5. The incident commander does not see the suspect [human physical activity]
7. The drone tracking the suspect will run out of battery in x minutes [battery replacement]
    a. A second UAV will takeoff to replace the one out of battery [drone replacement]

Use Case: Chemical accident
ID: UC-4

Description
Multiple UAV's are dispatched in order to analyse a gas source, observe its dissemination and help evacuating an area if required.

Primary Actor
Several UAVs

Supporting Actors
Firefighters

Stakeholders and Interests
Firefighters. Civilians.

Pre-Conditions
- Dronology system is active
- Multiple UAVs are equipped with cameras and are placed on the ground and are activated
- Multiple UAVs are equipped with multigas sensors (universal gas sensors)
- Multiplace UAVs are equipped with speficic gas sensors and weather sensors.
- Multiple UAVs are equipped with loudspeakers
- Firefighters have marked the accident area to be searched
- Search plan has been generated
- Critical infrastructures (hospitals, etc) have been identified and marked on DroneResponse
- DroneResponse is running and UAVs are displayed on map

- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition
The gas source has been identified. The dissemination is tracked successfully by the drones.
Failure end condition:
The gas source has not been identified or the dissemination is not tracked by drones.

Trigger
The Drone Commander activates the search.

Main Success Scenario

1. Emergency responders are dispatched to the scene of the accident.
2. Emergency responders release one drone to survey the accident.
3. The drone takes off.
4. The drone streams imagery to the DroneResponse UI.
5. The operator marks the scene of the accident.
6. The drone hovers in the air and continues to stream imagery of the accident.
7. The drone operator equips a small cohort of drones with appropriate gas sensors.
8. The drones coordinate their routes and start monitoring air quality.
9. Drones dynamically map out the region of the gas.
10. DroneResponse analyzes the gas map and visualizes the gas plume.
11. The operator identifies at-risk areas that could be impacted by the gas cloud.
12. The operator prepares a public voice message to warn the public.
13. DroneResponse assign routes to new drones in order to reach a critical infrastructure that has to be evacuated.
14. Drones equipped with loudspeakers takeoff.
15. Drones equipped with loudspeakers fly toward the assigned area.
16. Drones arrive at destination and use loudspeakers to alert civilians that they have to evacuate as fast as possible.[]
17. The incident commander recalls drones as deemed necessary.

Exceptions

3a. Drone fails to take off
    3a.1 Another drone takes off to replace the one that failed
4a. The drone fails to communicate with DroneResponse to stream images
    4a.1 The drone stores the entire video stream in its memory
    4a.2 The drone sends videos stored as soon as the connection gets back
11a. No critical infrastructure nearby
    11a.1 Drones continue observing the dissemination of the gas plume

---

Use Case: Fly to targeted location to sample water
ID: UC-8

Description
Multiple UAVs head to a specific area to sample water
Primary Actor
Drone Commander

Supporting Actors
Semi-autonomous UAV, Incident Commaner

Stakeholders and Interests
firefighters, chemical researchers

Pre-Conditions
- Dronology system is active
- Multiple UAVs are equipped with cameras, sampling systems and are placed on the active drones
- Firefighters/researchers have marked specific area to be sampled of the water quality
- Sample route has been generated
- DroneResponse is running and UAVs are displayed on map
- Each drone is able to carry multiple sampled containers
- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition
The samples are collected successfully by each UAV with its assigned water area.
Failure end condition:
Drones fails to collect samples back to home base

Trigger
The Drone Commander activates the search.

Main Success Scenario

1. DroneResponse is launched with a "water_sampling" mission.
2. Incident Commander marks the water places on the DroneResponse map.
3. DroneResponse calculates appropriate routes toward each area, avoiding prohibited areas.
4. DroneRespone assigns the routes previously calculated to each drone.
5. The UAVs takeoff and fly following their individually assigned routes.
6. After arriving at its targeted location, UAV switches to "sampling_mode".
7. After finishing its sampling mode, UAVs head to their next specific targeted water area (if there is any).
8. Repeats step 6 to 9 until water sampling for all the designated locations is completed.
9. Each drone returns back to its home-base to wait for researchers to collect the sample.

Exceptions

5a. One drone fails to take off
  5a.1 DroneResponse assigns the parameters of the drone to a new drone
  5a.2 The new drone takes off
7a. Drone arrives on scene and there is no water place
  7a.1 The Drone Commander is alerted through DroneResponse
  7a.2 The Drone Commander has to define a new place on DroneResponse map
  7a.3 New coordinates are transmitted to the drone
8a. UAV fails to fill the sampling container due to external factors
10a. UAV fails to head to next targeted water area due to battery
  10a.1 Remaining areas are assigned to the other flying drones or new drones takeoff if the battery of the flying ones is not sufficient to cover all areas

---

Use Case: Record Air quality level
ID: UC-9

Description
Multiple UAVs head to a specific area to sample air
Primary Actor
Drone Commander

Supporting Actors
Semi-autonomous UAV, Incident Commander

Stakeholders and Interests
Central Weather Bureau workers

Pre-Conditions
- Dronology system is active
- Multiple UAVs are equipped with cameras and are placed on the active drones
- Multiple UAVs are equipped with air-quality sensors
- Workers have marked specific area to be recorded of the air quality
- Sample route has been generated
- DroneResponse is running and UAVs are displayed on map
- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition
The air-quality data is sent successfully by each UAV with its assigned aerial area.
Failure end condition:
Drones fails to send back air-quality data

Trigger
The Drone Commander activates the search.

Main Success Scenario

1. DroneResponse is launched and the “air_sampling” mission is loaded by the Incident Commander.
2. The Incident Commander marks an area on DroneResponse map to analyze its air quality.
3. DroneResponse creates routes to reach the area by avoiding areas that are flight prohibited.
4. Each drone gets its route assigned.
5. The UAVs takeoff and head toward their assigned location.
6. Each UAV heads to its aerial area.
7. After arriving at its assigned coordinates, UAV switches to “recording_mode” where each drone records the air using its on-board air sensor, processes the air then sends the result to DroneResponse.
8. After finishing its recording_mode, UAVs head to their next specific targeted aerial area.
9. Repeat step 6 to 10 until each area to be covered have been done.
10. UAV’s then return to their home-based location.

Exceptions

5a. One drone fails to take off
- 5a.1 DroneResponse assigns the parameters of the drone to a new drone
- 5a.2 The new drone takes off

6a. Communication is lost with an UAV
- 6a.1 DroneResponse tries to restore communication every X seconds

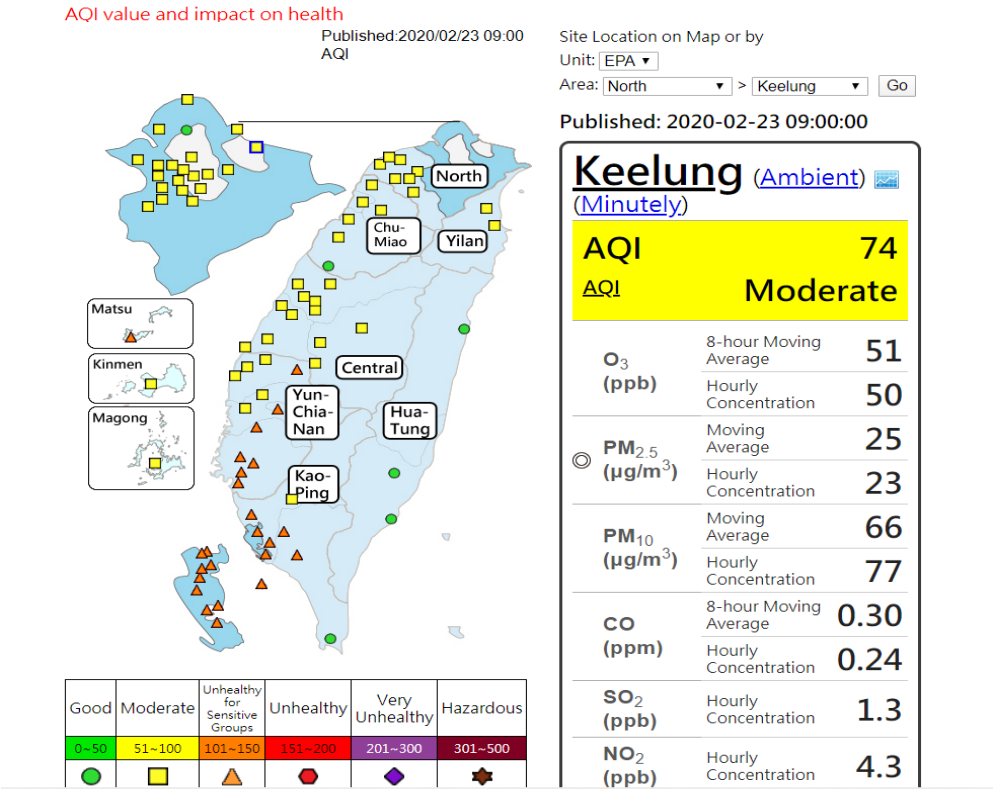
7a. Drone arrives on scene and there is no water place
- 7a.1 The Drone Commander is alerted through DroneResponse
- 7a.2 The Drone Commander has to define a new place on DroneResponse map
- 7a.3 New coordinates are transmitted to the drone

8a. UAV fails to recording air due to external factors

11a. UAV fails to head to next targeted water area due to battery

11a.1 Remaining areas are assigned to the other flying drones or new drones takeoff if the battery of the flying ones is not sufficient to cover all areas

Visual data window



Use Case: 3D restitution of a school
ID: UC-10

Description
Multiple UAV's are dispatched to build a 3D model of a school in order to identify possible entries and exits points for police to manage a shooting situation

Primary Actor
Several UAVs

Supporting Actors
Police. Firefighters. Medical crew.

Stakeholders and Interests

Pre-Conditions
- Dronology system is active
- Multiple UAVs are equipped with cameras and are placed on the ground and are activated
- Policemen have marked the accident area to be searched
- Search plan has been generated
- DroneResponse is running and UAVs are displayed on map
- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition
A 3D model of the school has been built.
Failure end condition:
A 3D model of the school has not been built.

Trigger
The Drone Commander activates the search.

Main Success Scenario

1. DroneResponse is launched and assigned the mission "building_restitution".
2. DroneResponse assign routes to the drones.
3. The drones takeoff and fly to the assigned location.
4. Arriving at the scene, one drone identifies the boundaries of the building.
5. Another drone starts looking for entry / exit doors of the building and marks each door found.
6. DroneResponse processes information from both drones and builds a 2D map containing the shape of the building and the marked entry/exit points.
7. Both drones now start gathering information to upgrade the existing model to a 3D map. They fly around the building within the boundaries previously identified.
8. DroneResponse synchronizes its 2D model with the new images given by the drones and builds the 3D model of the building.
9. Both drones switch to "advanced_search" mode and look for parts of the building the shooter might try to escape from.
10. Drones mark each new place found.
11. DroneResponse synchronizes its 3D model with the new places found.
12. DroneResponse now has access to a fully detailed model of the building and transmits information to the police.
13. The drone commander instructs the drones to return to launch.

Exceptions

1a. Failed to gather the coordinate because of a connection problem with Google Maps
    1a.1 DroneResponse uses another software to identify coordinates
3a. One drone fails to take off
    3a.1 DroneResponse assigns the parameters of the drone to a new drone
    3a.2 The new drone takes off
4a. Drones can't precisely identifie the boundaries of the building (Maybe buildings attached to each other?)
    4a.1 DroneResponse alerts the Drone Commander that drones are struggling to define precise boundaries of the building
    4a.2 The Drone Commander uses DroneResponse map to draw the boundaries of the building

Use Case: Man overboard

ID: UC-12

Description

Seaman has fallen overboard without an active locator

Primary Actor

Drone Commander

Supporting Actors

Semi-autonomous UAVs

Stakeholders and Interests

Naval officers

Pre-Conditions

- Dronology system is active
- Multiple search drones ready for dispatch
- Drone with flotation device on standby
- Victim fallen overboard

Post Conditions

Success end condition

The victim is rescued

Failure end condition:

The victim is not found or the victim is found but not actively tracked.

Trigger

A man-overboard alert is raised

Main Success Scenario

1. The search area is computed based on known currents, speed of the boat, and predicted time in the water.
2. A cohort of 3-4 drones is dispatched from the deck of the boat.
3. The drones self-organize to search the area.
4. DroneResponse actively tracks on drones in flight.
5. Drones use on-board image detection to identify a person in the water.
6. When the person is found, the GPS coordinates are immediately sent to DroneResponse.
7. The drones coordinate to ensure that the victim is tracked using line of sight.
8. A larger drone with a flotation device is immediately dispatched to the scene.
9. The flotation device is dropped close to the victim.
10. Human rescuers in a boat go directly to the scene to rescue the victim.
11. Once the victim is retrieved by the rescuers the drones determine the location of the ship deck using an active beacon.
12. The drones use the beacon signal to return to the location of the ship.
13. The drones use image recognition to identify the landing pad and land in the correct spot.

Use Case: Controlling crowd at sports event

ID: UC-13

Description

Swimmers must be cleared from the water for high-speed boat races in ocean

Primary Actor

Drone Commander

Supporting Actors

Semi-autonomous UAVs

Stakeholders and Interests

Pre-Conditions

- Dronology system is active
- Drones are equipped with loud speakers
- Race is about to start

Post Conditions

Success end condition

All swimmers have left the water

Failure end condition:

Swimmers remain in water preventing the start of the race

Trigger

Race is about to start

Main Success Scenario

1. Drones are dispatched from the takeoff site to fly along the shoreline.
2. The drones stream imagery back to the operator.
3. The drones use onboard vision capabilities to fly along the shoreline.
4. The drones search for swimmers in the water.
5. The drone detects a swimmer in the water.
6. The drone uses its onboard speaker system to tell the swimmer to get out of the water.
7. The drone continues to patrol along the beach in search of more swimmers.
8. When the drone has completed an initial sweep of its designated area it turns back.
9. The drone again checks for swimmers in the water.
10. The drone detects a swimmer who is still in the water.
11. The drone uses its onboard speaker system to tell the swimmer to get out of the water immediately.
12. When a swimmer refuses to get out of the water, the drone computes the GPS coordinates of the swimmer.
13. The drone transmits the GPS coordinates of the swimmer to the Beach Patrol police.
14. The drone hovers in the vicinity of the swimmer until the swimmer gets out of the water.
15. The drone completes its sweep along the shoreline.
16. The drone returns to base.

Use Case: Underwater ice rescue
ID: UC-14

Description
Child fall through ice and unable to get out

Primary Actor
Drone Commander

Supporting Actors
Semi-autonomous UAVs

Stakeholders and Interests

Pre-Conditions
- Dronology system is active
- Drone with rope and
- Victim trapped under the ice

Post Conditions
Success end condition
The victim is rescued
Failure end condition:
The drone is unable to place rope correctly

Trigger
Person under the ice in a lake

https://www.youtube.com/watch?v=KubDPkfRTIs

Main Success Scenario

1. A single drone is dispatched to search the pond.
2. The human operator marks the region of the lake in which the victim is suspected to have fallen through the ice on the map.
3. The drone constantly streams video back to the operator.
4. The imagery received by DroneResponse is annotated to show the victim and the rescuer as detected by the drone.
5. When the drone identifies a victim under the ice, it raises a victim found alert.
6. When the drone identifies a victim, it hovers overhead to provide a beacon for the victim's location.
7. The human rescuer slides over the ice to the location, cuts through the ice, and rescues the victim as per normal rescue operations.

8. The drone returns to its launch site.

** This one is kind of short.

Use Case: Supplies delivery during flood to people trapped on roofs of houses.
ID: UC-15

Description
https://www.fhwa.dot.gov/uas/resources/hif19019.pdf

Primary Actor

Supporting Actors
Semi-autonomous UAV

Stakeholders and Interests
Fire department engaged in river rescue
FAA concerned with flight regulations
General public

Pre-Conditions

- Dronology system is active
- Multiple UAVs are placed on the ground and are activated
- Recognition drones are equipped with cameras and loudspeakers
- Delivery drones are equipped with cameras and able to drop items with a parachute
- Firefighters have marked area of river to be searched
- Search plan has been generated
- DroneResponse is running and UAVs are displayed on map
- A victim is in the search area
- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition

Failure end condition:

Trigger
The Drone Commander activates the search.

Main Success Scenario

1. DroneResponse is launched with "delivery_supplying" mission.
2. The flooded area to cover is marked on DroneReponse UI.
3. DroneReponse generates a search plan and determines how many drones are required for the house inspection process.
4. The search drones receive a specific search plan from DroneResponse.
5. The search drones take off.
6. The search drones fly toward the flooded area to be covered.
7. The search drones arrive on site and switch to "inspect_mode".
8. Each search drone starts by identifying all the houses in its assigned area.
9. Then, each search drone inspects every house identified in order to find people trapped on the roof.
10. When a drone finds people trapped on the roof, it lowers it's altitude and gets closer to the people.
11. The search drone maintains the safety distance with the people on the roof.
12. The search drone uses its loudspeakers to inform people to stay safe on the roof and that deliveries are on their way.
13. DroneResponse assigns the coordinates of the house to a delivery drone.
14. The delivery drone takes off.
15. The delivery drone reaches the assigned house.
16. The delivery drone locates the best spot to deliver the package.
17. The delivery drone drops the package.
18. The delivery drone returns to its home-based location.

19. The search drone continues its inspection process to find more people.
20. When the entire area has been covered, the search drones request the operator for a landing spot.

Use Case: Swimmer in rip current
ID: UC-17

Description
A swimmer is caught in a rip current and is being swept out to sea

Primary Actor
Drone Commander

Supporting Actors
Semi-autonomous UAVs

Stakeholders and Interests
Life-Guards
Coast-Guards

Pre-Conditions
- Dronology system is active
- Multiple search drones ready for dispatch

Post Conditions
Success end condition
The victim is rescued
Failure end condition:
The victim is not found or the victim is found but not actively tracked.

Trigger
Swimmer in rip-current alert sounded

Main Success Scenario

1. Swimmers on the beach raise the alarm that somebody is caught in a rip current.
2. Life guards notify the coast guard.
3. Life guards are unable to visually locate the victim.
4. The coast guard dispatches rescue boats to the scene to search the area.
5. The coast guard dispatches multiple drones to the scene.
6. The drones immediately divide up the search area and start searching for the victim.
7. A drone locates the victim with low degree of certainty given the waves.
8. The drone attempts to track the victim.
9. The drone streams video to the DroneResponse UI.
10. A human responder confirms the potential sighting.
11. The coast-guard sends the rescue boat to the coordinates of the drone.
12. DroneResponse dispatches a specialized drone carrying a flotation device to the victim.
13. The drone drops the flotation device near the victim.
14. The drone attempts to position itself as a beacon indirectly above the victim.
15. The operator instructs the drone to continue tracking.
16. The rescuers reach the victim and rescue him.
17. All drones return to launch

Use Case: Search a long section of a river for a lost-kayaker

ID: UC-18

Description

Two UAVs search each bank of the river in a downstream direction for the kayaker

Primary Actor

Drone Commander

Supporting Actors

Semi-autonomous UAV, Incident Commaner

Stakeholders and Interests

Emer

Pre-Conditions

- Kayaker has been reported missing
- Multiple UAVs are equipped with visible imagery cameras
- Area of the river has been marked
- All UAVs are equipped with collision avoidance technology
- Image detection trained to locate a kayak as well as a person

Post Conditions

Success end condition

The kayaker is rescued or a body is retrieved from the river

Failure end condition:

Neither the kayak nor the kayaker is not found

Trigger

The Drone Commander activates the search.

Main Success Scenario

1. Emergency responders go to a starting point on the river with two drones.
2. The drone operator identifies the next landing spot on the map.
3. The coordinates of the next landing spot are uploaded to the drones.
4. The drones are each assigned one of the river banks.
5. The drones take off.
6. The drones fly along the river banks.
7. The drones use onboard visible imagery cameras to search for a kayak or person.
8. The drones stream imagery to DroneResponse
9. Imagery is displayed on the UI
10. When the drones reach the vicinity of the next landing spot they hover in place until instructed to land.
11. The drone operator instructs each drone to land in turn.
12. The drone operator changes the drone's batteries.
13. Steps 4 to 10 are repeated until either the entire river section has been searched or the victim has been found.
14. The drone finds a kayak or the victim caught in tree branches and raises an alert.
15. The drone hovers in place awaiting further instructions.
16. The human operator checks the imagery and confirms that a victim has been sighted.
17. The GPS coordinates are sent to drone response.
18. A rescue (or retrieval) boat is sent to the location.
19. When the rescue boat arrives, the drones return to a location marked by the drone operator.

Use Case: Inspecting inaccessible houses after volcanic eruption
ID: UC-19

Description

Primary Actor

Supporting Actors
Semi-autonomous UAV

Stakeholders and Interests
Fire department engaged in river rescue
FAA concerned with flight regulations
General public

Pre-Conditions

- Dronology system is active
- Multiple UAVs are equipped with cameras and are placed on the yyund and are activated
- Firefighters have marked area of river to be searched
- Search plan has been generated
- DroneResponse is running and UAVs are displayed on map
- A victim is in the search area
- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition

Failure end condition:

Trigger
The Drone Commander activates the search.

Main Success Scenario

1. The drone commander assigns “disaster_inspection” mission on DroneResponse.
2. The drone commander inputs coordinates to draw the search area on DroneResponse map.
3. DroneResponse processes the coordinates.
4. DroneResponse calculates a search plan so that the defined area is fully covered and each drone receives an equivalent workload.
5. DroneResponse decides how many drones are required to search the area efficiently.
6. The mission plan is loaded into each drone.
7. The drones takeoff.
8. The drones fly toward their assigned areas.
9. Each drone activates its camera and starts streaming images live on DroneResponse UI.
10. Each drone starts to identify the houses in their assigned area.
11. Each drone flies around every house of its research area in order to identify the damages.
12. DroneReponse collects video imagery from multiple angles.
13. DroneResponse streams video imagery to a user interface.
14. The operator requests more details for a damaged part of the house.
15. The drone moves to the requested area.
16. The drone streams video imagery.
17. Once each drone has inspected every home in its area, it returns to its launch site.

Use Case: Inspecting utilities after a major storm
ID: UC-20

Description
https://www.precisionhawk.com/blog/drone-imagery-increases-speed-and-safety-of-repairs

Primary Actor

Supporting Actors
Semi-autonomous UAV

Stakeholders and Interests
Fire department engaged in river rescue
FAA concerned with flight regulations
General public

Pre-Conditions
- Dronology system is active
- Multiple UAVs are equipped with cameras and are placed on the yyund and are activated
- Firefighters have marked area of river to be searched
- Search plan has been generated
- DroneResponse is running and UAVs are displayed on map
- A victim is in the search area
- All UAVs are equipped with collision avoidance technology

Post Conditions
Success end condition

Failure end condition:

Trigger
The Drone Commander activates the search.

Main Success Scenario

1. The drone commander assigns “utility_impact_assessment” mission on DroneResponse.
2. The drone commander inputs the location of the mission.
3. DroneResponse assigns parameters to the drone, including flight regulations and collision avoidance.
4. The drone takes off and flies toward the utility location.
5. Arrived on site, the drone has to adequately target the utility, by identifying its boundaries.
6. Then, the drone provides real-time images of the utility state.
7. Following, the drone looks for the best access point for the repair crews to reach the utility by evaluating the surroundings.
8. The drone finally focuses on the utility itself and the damages so that the repair crews can prepare the right material before going on site.
9. DroneResponse streams the imagery of the drone camera on its UI.
10. The Drone Commander informs the repair crew about the current situation.

Use Case: Map an area after earthquake
ID: UC-16

Description
An earthquake has hit a major city and drones are used to map the area.
https://blog.dronedeploy.com/after-mexico-city-earthquake-drones-help-make-sense-of-the-damage-c056ce1d486c

Primary Actor
Drone Commander

Supporting Actors
Semi-autonomous UAVs
Urban Planners

Stakeholders and Interests
Public
Architects
Building inspectors

Pre-Conditions
- Dronology system is active
- Multiple drones

Post Conditions
Success end condition
The area is accurately mapped in a timely fashion

Failure end condition:
Areas with significant damage are missed

Trigger
Urban planners request aerial mapping following an earthquake

Main Success Scenario

1. The urban planners select an area of the city to be mapped.
2. DroneResponse operators prepare drones for the mission.
3. DroneResponse divides the area into segments representing a mapping area.
4. DroneResponse assigns each drone a distinct mapping area.
5. Each drone is equipped with visible camera capabilities.
6. Each drone plans flight routes to provide coverage of its assigned area.
7. Drones take off and fly to their assigned mapping area.
8. When the drone arrives at its assigned mapping area it activates cameras.
9. The drone collects and stores imagery.
10. The GPS coordinates of the drone and configuration of the camera is recorded regularly.
11. The drone returns to its base and lands.
12. After landing, the drone uploads its imagery to a server for processing.
13. DroneResponse reconstructs a single visual map of the area from multiple video streams and geolocates imagery onto a map.

Exceptions:
Battery replacement
Bad weather